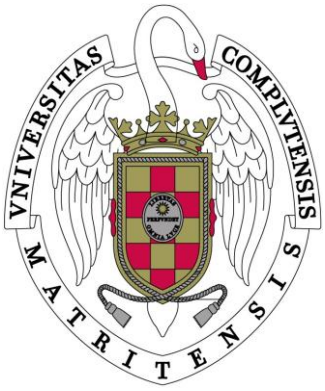

Sistemas Informáticos
2008/2009

Diseño de un simulador para redes de sensores

Alumnos:
Alejandro Asensio González
Arturo Miguel Núñez
Javier Pascual García

Profesor director:
Marcos Sánchez-Elez Martín

Facultad de Informática

Universidad Complutense de Madrid

TÍTULO:

Diseño de un simulador para redes de sensores

AUTORES:

Asensio González, Alejandro

Miguel Núñez, Arturo

Pascual García, Javier

PROFESOR DIRECTOR:

Sánchez-Elez Martin, Marcos

CURSO ACADÉMICO:

2008-2009

ASIGNATURA:

Sistemas Informáticos

Resumen

Uno de los puntos clave en las aplicaciones biomédicas actuales es el desarrollo de una nueva generación de dispositivo para la asistencia a personas con discapacidades, enfermedades crónicas o necesidades especiales para mejorar su autonomía e independencia. Este objetivo está cobrando un mayor relieve dado el envejecimiento de la población en los países desarrollados. La falta de recursos humanos para atender este problema conduce de forma natural al desarrollo de sistemas capaces de hacerlo de forma autónoma.

Para llevar a cabo esta tarea es importante la depuración, tanto de dispositivos como de circuitos, y ello conlleva simular en entornos informáticos el sistema y el modelo que se quiere implementar.

Debido a que no hay simuladores que modelen el comportamiento de nuestro nodo, nuestra labor ha estado centrada principalmente en la adaptación del simulador MSPSim y en la modificación de la herramienta COOJA de simulación de redes de sensores para nodos simulados bajo la herramienta MSPSim.

Después de nuestras modificaciones, ambas herramientas están preparadas para mostrar consumos de potencia fiables y paquetes enviados y recibidos por la radio de cada nodo. Asimismo se ha permitido alterar el tamaño de la memoria del chip y de la radio en MSPSim para simular programas de mayor envergadura.

Overview

One of the important keys of today's health care applications is the development of a new generation of assist tools for people with disabilities, chronic patients o specials needed in order to improve their autonomy and independency. This objective is getting important because of the aging of the population in developed countries. The lack of human resources to solve this problem guides naturally the development of systems that can do it automatically.

In order to do this task the experimentation is so important, that make us simulate at computer environment the system and the model we want to deploy on real nodes.

Because of there are not simulators that fix all the characteristics of our node, our work has been focused mainly in the adaptation of MSPSim simulator and the modification of the sensor network tool, COOJA, which is ready to work with MSPSim directly.

After our modifications, both tools are ready to show power consumption information with a high level of accuracy and packets sent and received by the radio of each node. Likewise It's possible to change the memory size of the chip and radio in MSPSim in order to simulate bigger programs.

Índice

1. Introducción

En la actualidad la esperanza de vida de la población esta creciendo enormemente, esto conlleva el envejecimiento de la población y el aumento de costes del diagnostico de enfermedades. Para reducir estos costes se propone realizar diagnósticos de forma automática. No todas las enfermedades pueden diagnosticarse sin necesidad de un experto pero por el contrario existen otras en las que sí es factible.

Nuestro proyecto esta enmarcado en un proyecto mas ambicioso como es el de diagnosticar enfermedades cardiacas entre otras, a través de sensores que captan señales del organismo y las interpretan. A cada uno de los sensores que forman la red se le denomina nodo. Para conseguirlo se conectarán varios nodos entre si para captar mayores síntomas y ampliar las enfermedades a diagnosticar.

Dentro de estos nodos se pueden ejecutar distintas aplicaciones tales como monitorización del medio ambiente, cuidado de la salud, control del tráfico, detección de incendios o seísmos, monitorización de deportistas, etc.

En Redes de Sensores Inalámbricas (WSN), como las mostradas en la figura 3.2, docenas, cientos o incluso miles de sistemas diminutos, a pilas, sistemas empotrados de tiempo real son dispersados en todas partes de un ambiente físico. Cada nodo forma parte de una red *ad-hoc* que recoge datos, como la temperatura, la humedad, la vibración u otros factores exógenos de interés. El nodo retransmite los datos recogidos a una estación base donde son procesados para proporcionar una imagen fiel del entorno en tiempo real.

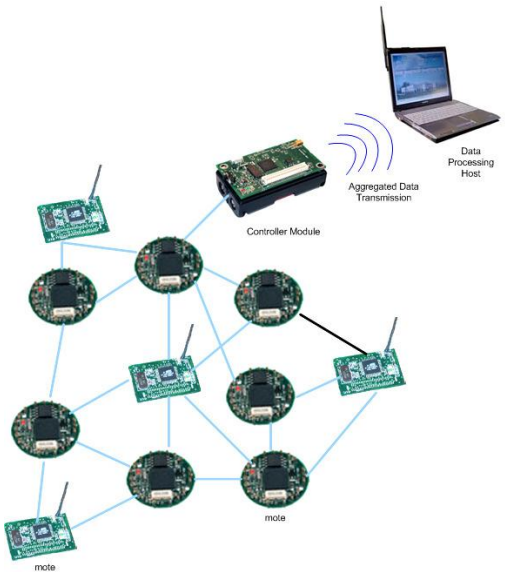

Figura 3.2: Red de Sensores *ad-hoc*

Todo proyecto de estas características, proyectos de investigación, requieren un alto nivel de experimentación, para ello es importante el uso de simuladores para evitar cargar de trabajo innecesario a los sensores o incluso dañarlos. Con este tipo de herramientas podemos obtener datos empíricos de entre otras cosas, uso de CPU del chip, uso de la memoria, rendimiento etc. pero sobretodo poder obtener consumos de energía del sensor ya que al ser nodos sin alimentación externa se debe optimizar el uso de la energía. Tan importante como obtener datos empíricos es realizar ciertas pruebas con entornos distintos al real para averiguar si alguna modificación en el nodo pudiera optimizar otros recursos del mismo.

Existen algunas herramientas para simular tanto sensores, como redes de sensores, pero ninguna se adaptaba totalmente a nuestro sensor, por lo que ha sido necesario realizar modificaciones a la herramienta que más se asemejaba. Las herramientas escogidas finalmente fueron MSPSim como nuestro simulador a nivel individual del nodo y COOJA como simulador de redes de sensores. COOJA simula nativamente los nodos simulador por MSPSim facilitando enormemente la tarea de integración de ambas herramientas.

Una vez simulada la red de sensores es importante monitorizar el tráfico de la radio entre nodos, este es otro de los aspectos importantes de nuestro proyecto. Ver como afecta la topología de la red al envío/recepción de paquetes y como afecta esto al consumo de potencia.

Con estas premisas en mente hemos realizado el trabajo que a continuación se expone.

2. Redes de sensores

Una red de sensores es una red de nodos, que colaboran en una tarea común. Estos nodos poseen ciertas capacidades sensitivas y de comunicación inalámbrica que les permite formar redes *ad-hoc [1],* es decir sin infraestructura física preestablecida ni administración central.

Esta clase de redes se caracterizan por su facilidad de despliegue y por ser autoconfigurables, pudiendo convertirse en todo momento en emisor o receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo.

Uno de sus principales problemas es la gestión eficiente de la energía de forma que se consiga obtener una alta tasa de autonomía de forma que sean plenamente operativas.

2.1. Áreas de aplicación

El ámbito de aplicación de las redes de sensores es muy amplio y diverso, en particular cabe destacar las siguientes áreas de aplicación

- **Medicina**: Con la reducción de tamaño que están sufriendo los nodos sensores, la calidad de vida de pacientes que tengan que tener controlada sus constantes vitales (pulsaciones, presión arterial, tensión, nivel de azúcar en sangre, etc.), podrá mejorar substancialmente.

- **Entornos de alta seguridad**: Existen lugares que requieren altos niveles de seguridad para evitar ataques terroristas, tales como centrales nucleares, aeropuertos, edificios del gobierno de paso restringido. Aquí gracias a una red de sensores se pueden detectar situaciones que con una simple cámara sería imposible.

- **Sensores ambientales**: El control ambiental de vastas áreas de bosque o de océano, sería imposible sin las redes de sensores. El control de múltiples variables, como temperatura, humedad, fuego, actividad sísmica así como otras. También ayudan a expertos a diagnosticar o prevenir un problema o urgencia y además minimiza el impacto ambiental de la presencia humana.

- **Sensores industriales:** Dentro de fábricas existen complejos sistemas de control de calidad, el tamaño de estos sensores les permite estar allí donde se requiera.

- **Automoción**: Las redes de sensores son el complemento ideal a las cámaras de tráfico, ya que pueden informar de la situación del tráfico en ángulos muertos que no cubren las cámaras; también pueden informar a conductores de la situación, en caso de atasco o accidente, con lo que estos tienen capacidad de reacción para tomar rutas alternativas.

- **Domótica:** Su tamaño, economía y velocidad de despliegue, las hacen una tecnología ideal para automatizar tareas cotidianas en el hogar a un precio asequible.

2.2. Características

Las redes de sensores tienen una serie de características propias y otras adaptadas de las redes *ad-hoc*:

- **Topología Dinámica**: En una red de sensores, la topología siempre es cambiante, debido a que los nodos pueden dejar de funcionar en cualquier momento, y éstos tienen que adaptarse para poder comunicar nuevos datos adquiridos.

- **Variabilidad del canal:** El canal de radio es un canal muy variable en el que existen una serie de fenómenos como pueden ser la atenuación, desvanecimientos rápidos, desvanecimientos lentos e interferencias que puede producir errores en los datos.

- **No se utiliza infraestructura de red**: Una red de sensores no tiene necesidad alguna de infraestructura para poder operar, ya que sus nodos pueden actuar de emisores, receptores o *router*[1].

  Sin embargo, hay que destacar en el concepto de red de sensores la figura del nodo recolector (también denominados *sink node*), que es el nodo que recopila la información generada normalmente en tiempo discreto. Esta información generalmente es adquirida por un ordenador conectado a este nodo y es sobre el ordenador donde recae la posibilidad de transmitir los datos por tecnologías inalámbricas o cableadas según sea el caso.

- **Tolerancia a errores**: Un dispositivo sensor dentro de una red de sensores tiene que ser capaz de seguir funcionando a pesar de tener errores en el sistema propio.

- **Comunicaciones multisalto o broadcast**: En aplicaciones de redes de sensores siempre es característico el uso de algún protocolo que permita comunicaciones *multi-hop* [2], aunque también es muy común utilizar mensajería basada en *broadcast*.

- **Consumo energético**: Es uno de los factores más sensibles debido a que tienen que conjugar autonomía con capacidad de proceso, ya que actualmente cuentan con una unidad de energía limitada. Un nodo sensor tiene que contar con un procesador de consumo ultra bajo así como de un transceptor radio con la misma característica, a esto

[1] Router es un dispositivo de hardware para interconexión de red de ordenadores que opera en la capa de red. Determina la ruta que debe tomar el paquete de datos.

hay que agregar un software que también conjugue esta característica haciendo el consumo aún más restrictivo.

- **Limitaciones hardware**: Para poder conseguir un consumo ajustado, se hace indispensable que el hardware sea lo más sencillo posible, así como su transceptor de radio, esto nos deja una capacidad de proceso limitada. Por ejemplo la radio utilizada tiene que emplear un protocolo de bajo consumo de energía

- **Costes de producción**: Dado que la naturaleza de una red de sensores tiene que ser en número muy elevada, para poder obtener datos con fiabilidad. Una vez definida su aplicación, los nodos sensores son económicos de hacer si éstos se producen en grandes cantidades.

2.3 Tecnología de radio IEEE 802.15.4

El estándar IEEE 802.15.4 [3], cuya última revisión se aprobó en 2006, fue creado para cubrir la necesidad del mercado de estándares inalámbricos de baja tasa para aplicaciones en redes de sensores. Los estándares existentes hasta el momento en el mercado estaban destinados a aplicaciones con mayores requisitos en cuanto a ancho de banda se refiere, como pueden ser videoconferencias o redes domésticas. Los ejemplos más representativos de estas tendencias son el IEEE 802.11[4], también conocido como Wi-Fi y el IEEE 802.15.1[5] conocido como Bluetooth.

Los inconvenientes que surgían al utilizar cualquiera de éstos estándares, eran su gran consumo de energía y ancho de banda frente a la baja tasa y bajos requisitos de energía necesaria para las redes de sensores.

A continuación se puede observar una pequeña comparativa entre el estándar 802.15.4 y otros estándares como son *Bluetooth* y *Wi-Fi*:

Estándar	Ancho de banda	Consumo de potencia	Ventajas	Aplicaciones
Wi-Fi	Hasta 54 Mbps	400 mA transmitiendo 20 mA en reposo	Gran ancho de banda	Navegar por Internet, redes de ordenadores transferencia de ficheros
Bluetooth	1 Mbps	40 mA transmitiendo 0.2mA en reposo	Interoperatividad, sustituto del cable	Wireless USB, móviles, informática doméstica
802.15.4	250 Kbps	1,8 mA transmitiendo 5,1 µA en reposo	Batería de larga duración, bajo coste	Control remoto, productos dependientes de la batería, sensores etc.

Figura 2.1: Comparativa estándares Wireless.

La tecnología inalámbrica 802.15.4 permite comunicaciones de corto alcance con distancias de hasta 75m y bajo consumo. Puede funcionar en la banda de 2,4 GHz a una tasa de 250 Kbps, en la de 868 MHz a 40 Kbps y en la de 915 MHz a 20 Kbps, aunque la mayoría de fabricantes optarán por la elección de la primera ya que puede ser usada en todo el mundo, mientras que las dos últimas sólo se pueden usar en Europa y EEUU, respectivamente.

El objetivo es que un sensor equipado con esta tecnología pueda ser alimentado con dos pilas AA un periodo de entre seis meses y dos años, dependiendo del tipo de aplicación.

2.3.1. Mecanismos de robustez.

El estándar IEEE 802.15.4 ha sido desarrollado para entornos hostiles y medios compartidos en los que puede ocurrir que haya colisiones de tramas, que exista ruido o que las tramas no lleguen correctamente, por lo que se han definido una serie de mecanismos para que sea robusto:

- **CSMA-CA**: sistema basado en la detección de portadora evitando colisiones. Su esquema de funcionamiento lo hace excelente compartiendo el medio.

- **Tramas con confirmación (ACK):** cuando enviamos tramas, se nos devuelve una trama ACK confirmando que la trama de datos o cualquier otra ha sido recibido correctamente.

- **Verificación de los datos (CRC)**: mediante un polinomio generador de grado 16 se obtiene la redundancia y se puede comparar el CRC enviado con el calculado en el destino a fin de verificar que el dato se haya transmitido correctamente.

- **Restricciones de consumo:** IEEE 802.15.4 está pensado para aplicaciones que utilicen una batería o una unidad de energía agotable. Estas aplicaciones transmitirán información de forma muy esporádica por lo que la cantidad de energía que consume un nodo cuando escucha el canal es muy baja.

- **Seguridad**: implementa seguridad de clave simétrica mediante el estándar de encriptación AES[2]. El manejo y gestión de la claves es derivado a capas superiores.

[2] AES es un esquema de cifrado por bloque adoptado como un estándar de encriptación por el gobierno de los Estados Unidos, y se espera que sea usado en el mundo entero.

En conclusión, IEEE 802.15.4 resulta ideal para redes de sensores, escalables, con muchos dispositivos, pocos requisitos de ancho de banda y dónde se requiera una duración muy prolongada de la batería. Por lo tanto, se puede afirmar que en ciertas condiciones y para determinadas aplicaciones puede ser una buena alternativa a otras tecnologías inalámbricas ya consolidadas en el mercado.

3. Plataforma SHIMMER

SHIMMER [4] es el acrónimo de *Sensing Health with Intelligence, Modularity, Mobility and Experimental Reusability*.

Es una pequeña plataforma de sensores inalámbricos diseñada por Intel para aplicaciones portátiles de cuidado de la salud.

El objetivo de SHIMMER es proporcionar un sistema independiente a aquellos pacientes que necesitan un control diario de su salud, de forma que puedan seguir con una vida normal en sus casas.

Con esto en mente se ha diseñado el concepto de *body sensor network* (BSN) en el cual sensores individuales conectados, normalmente vía inalámbrica mediante protocolos tales como 802.15.4, proporcionan una visión en conjunto del estado de bienestar de una persona. Una ilustración de este concepto se puede ver en la siguiente figura.

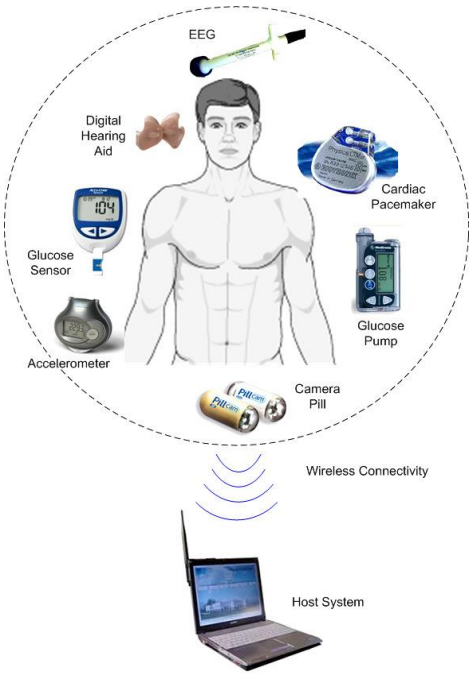

Figura 3.1: *Body Area Sensor Network*

Un Nodo o *Mote* es una plataforma hardware con capacidad de procesamiento y de comunicación. Los sensores se conectan mediante la radio al procesador del Nodo, está combinación le confiere al nodo las habilidades de sentir, computar y comunicar.

Las redes de sensores puede ser utilizadas para monitorizar la salud de las personas, donde los sensores inalámbricos están estratégicamente colocados en distintas partes del cuerpo, pueden monitorizar distintas constantes vitales, proporcionar datos en tiempo real al usuario y al personal médico.

3.1 Características

Algunas de las características principales de las redes de sensores para aplicaciones médicas son la fiabilidad, la biocompatibilidad, la portabilidad, la privacidad y la seguridad.

Estas aplicaciones requieren una capacidad de cómputo más pequeña que las ejecutadas por una PDA, por ejemplo. Se ejecutan en dispositivos de bajo consumo de potencia, comunicación inalámbrica y batería de larga vida.

Dichas aplicaciones se puede encargar de realizar electrocardiogramas, tomar el pulso, medir la saturación de oxígeno en sangre, controlar la respiración, la temperatura de la piel o la presión arterial

3.2 Componentes

Los principales componentes del dispositivo SHIMMER son los siguientes:

- El procesador: 8MHz Texas Instruments™ MSP430 CPU (10Kbyte RAM, 48Kbyte Flash, 8 Channels of 12 bit A/D)
- Radio Clase 2 Bluetooth
- 2.4GHz IEEE 802.15.4 Chipcon™ wireless transceiver
- 3-Axis Freescale™ accelerometer
- Ranura MicroSD con capacidad para tarjetas de hasta 2 Gbytes
- Bateria Li-Ion integrada

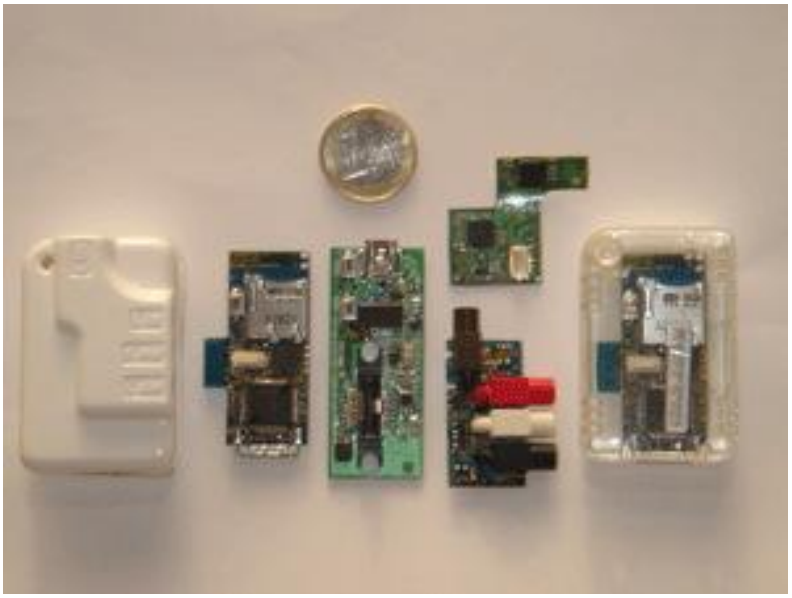

Figura 3.3: SHIMMER con varios accesorios y carcasas

3.3 Arquitectura

En esta sección se van a describir los distintos subsistemas y componentes hardware de la arquitectura de la plataforma SHIMMER. Se pretende poner de manifiesto el potencial de la plataforma para aplicaciones inalámbricas en el contexto del cuidado de la salud.

El elemento central de la plataforma es el microprocesador MSP430 [6] de bajo consumo de potencia que controla el funcionamiento del dispositivo. La CPU se comunica con diversos periféricos a través de los módulos internos y externos de ampliación. Captura los datos desde el conversor analógico-digital de 8 canales.

El acelerómetro de XYZ permite la lectura de aceleración 3-dimensiones. La expansión interna permite la conexión de sensores adicionales, tales como el electrocardiograma, la cinemática, electromiograma y el electroencefalograma.

La expansión permite la comunicación desde y hacia la placa base utilizando la estación de acoplamiento.

La placa del SHIMMER tiene una ranura MicroSD Flash para almacenamiento adicional y tiene cuatro diodos (LED) para la visualización. Dispone de dos radios, BlueTooth y 802.15.4 para el envío de datos

La figura 4.3 ilustra un diagrama de bloques de la placa base del SHIMMER y de sus interconexiones entre los dispositivos integrados

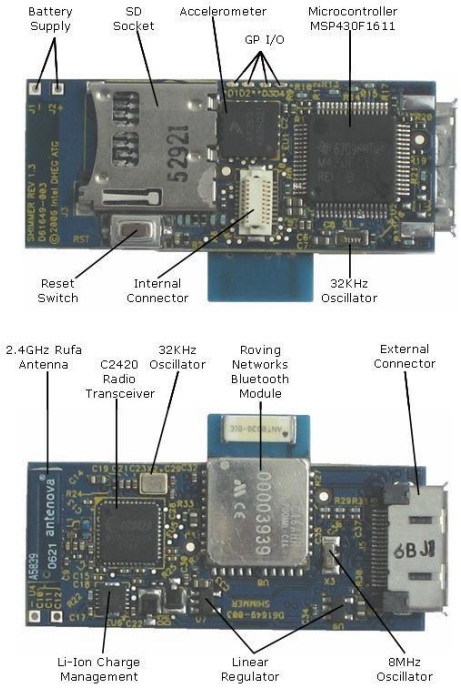

Figura 3.4: Fotografía de la placa base de SHIMMER

3.3.1 CPU

La principal ventaja del MSP430 es su bajísimo consumo de potencia durante los períodos de inactividad.

El MSP430 se basa en una arquitectura RISC de 16-bit, periféricos y un sistema de reloj que se combinan con un direccionamiento Von Neumann para bus de memoria (MAB) y bus de datos (MDB).

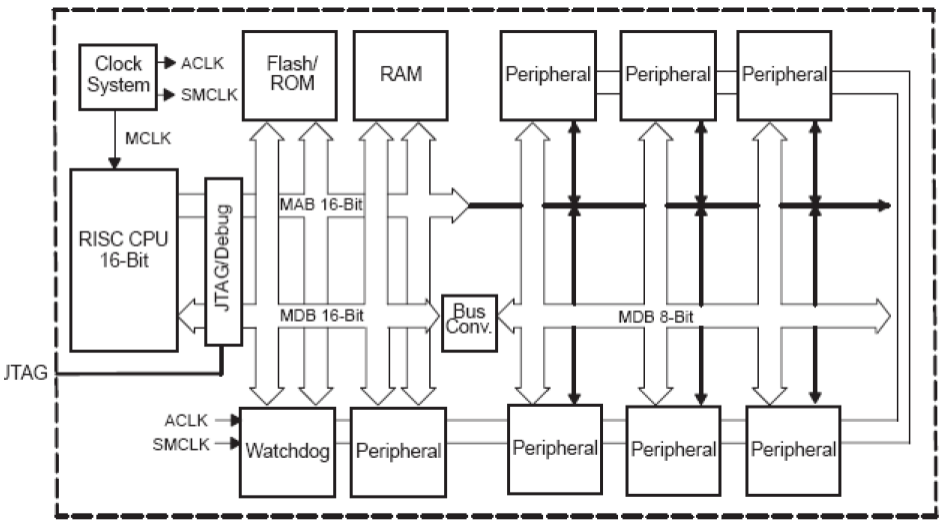

Figura 3.5: Diseño arquitectónico del MSP430

La CPU usa una arquitectura von Neumann, con direccionamiento simple para las instrucciones y los datos. La memoria se direcciona por byte, y los pares de byte se combinan en forma endianness[3] para hacer palabras de 16 bits.

El procesador contiene 16 registros de 16 bits. R0 es el contador de programa, R1 es el puntero de pila, R2 es el registro de estado, y R3 es un registro especial denominado generador constante, que provee acceso a 6 valores constantes comúnmente utilizados sin requerir un operando adicional. Desde R4 hasta R15 se encuentran los registros de uso general.

El conjunto de instrucciones es muy simple; hay 27 instrucciones agrupadas en tres familias. La mayoría de las instrucciones son de 8 bits (un byte) y 16 bits (palabra), dependiendo del valor de un bit llamado B/W, este bit es 1 para instrucciones de 8 bits y 0 para las de 16 bits. Las operaciones de byte sobre la memoria afectan solo al byte direccionado, mientras que las operaciones de byte sobre los registros al byte más significativo.

Véase anexo B: Repertorio de instrucciones MSP430

3.3.1.1 Ciclo de reloj

El mecanismo de reloj del MSP430 está diseñado para soportar un consumo bajo de batería permitiendo al usuario seleccionar entre tres señales con el objetivo de obtener una correlación optima entre el rendimiento y el consumo de potencia. Las tres señales de reloj son:

- *Auxiliary Clock* (ACLK): Es una señal de reloj de baja frecuencia suministrada por un cristal de 32kHz utilizado para el modo de consumo de potencia bajo y las para las operaciones de *self wake-up*[4] *y self-timing*[5].

- Master Clock (MCLK): Señal de reloj de alta velocidad que se alimenta desde un DCO (*digitally controlled oscillator*) usando solicitudes de interrupción. Se usa para la señal de alta velocidad de CPU y para las aplicaciones de periféricos. El oscilador puede operar hasta 8MHz, puede ser activada desde el modo sleep en 6 µs, en el cual el dispositivo está en un modo de reposo de forma que realiza un muy bajo consumo potencia, y permitir a la CPU estar operativa en arranques muy rápidos.

[3] Endianess - hace referencia a la forma de almacenar los bits en memoria.
[4] Self wake-up - autoarrancado.
[5] Self-timming - automuestreo.

- Sub-Main Clock (SMCLK): Señal diseñada para utilizar el DCO u osciladores estándar que son necesarios para las operaciones con periféricos.

3.3.1.2 Memoria

El MSP430 tiene 10Kbytes de RAM, 48Kbytes de Flash y 128bytes de almacenamiento de información. La memoria flash puede programarse completamente en el sistema usando la CPU, el bootstrap loader o el puerto JTAG.

La memoria flash puede almacenar tanto datos como código y la CPU puede escribir en ella mediante *single-bytes y single-words.*

3.3.1.3 Timmers

El microprocesador MSP430 está equipado con dos *temporizadores,* Timer_A y Timer_B.

Timer_A es un timmer/counter asíncrono de 16 bits que contiene tres registros *capture/compare* que permiten *salidas PWM, intervalo de muestreo* y múltiples *capture/compare.* Timer_A también incluye la entrada asincrónica, la salida *latching*[6] y la capacidad de generar interrupciones dependiendo de las condiciones de desbordamiento de los registros *capture/compare.*

El Timer_B es idéntico al Timer_A con la funcionalidad adicional de tener una longitud programable de 8, 10, 12, o 16 bits, doble buffer y registros de grupo capaces de forzar la salida en un estado de impedancia alta. Timer_B sin embargo, no contiene el bit de capture/compare sincronizado.

3.3.1.4 USARTs

El MSP430 contiene dos puertos serie de comunicación (USART0 y USART1). Se puede enviar y recibir datos simultáneamente usando los canales de doble buffer.

3.3.1.5 ADCs

Tiene 8 canales ADC[7] para conversiones de Analógico/Digital de 12 bits que usan un buffer que permite leer y almacenar datos sin la intervención de la CPU.

[6] Latching - guardar el estado.
[7] ADC – Analogue-to-digital converter

Los puertos externos son utilizados para leer datos del acelerómetro XYZ (3 canales), el conector de extensión interno (3 canales) y el conector de extensión externo (2 canales). Los puertos ADC internos pueden ser usados para leer datos del sensor interno de temperatura o leer el voltaje de la batería.

La tasa máxima de conversión del ADC es superior a 200Ksps y el período de muestreo puede ser controlado por el software o por uno de los temporizadores internos.

Con el objetivo de obtener un consumo de potencia bajo, el conversor analógico digital está deshabilitado cuando no está un uso y se rehabilita cuando es necesario.

3.3.1.6 Modos de ahorro de energía

Para obtener un compromiso entre procesamiento y consumo de potencia, MSP430 viene provisto de un modo activo y de cinco modos seleccionables de bajo consumo de potencia. Utilizando las tres distintas señales se obtienen los seis modos de ejecución.

Modes	CPU	MCLK	SMCLK	DCO's DC generator	ACLK	Current draw (µA) at 3V, 1MHz
Active mode (AM)	1	1	1	1	1	340
Low-power mode 0 (LPM0)	0	0	1	1	1	70
Low-power mode 1 (LPM1)	0	0	1	1	1	70
Low-power mode 2 (LPM2)	0	0	0	0	1	17
Low-power mode 3 (LPM3)	0	0	0	0	1	2
Low-power mode 4 (LPM4)	0	0	0	0	0	0.1

1 : enabled, 0 : disabled

Figura 3.6: Modos de ejecución de MSP430.

La figura 3.6 muestra los distintos modos de ejecución en modo ahorro de energía en base a las distintas señales de reloj.

3.3.2 Entrada/Salida

La plataforma SHIMMER contiene varios dispositivos de E/S que incrementan la funcionalidad del dispositivo.

3.3.2.1 Acelerómetro

El Freescale SemiconductorTM 3-axis (XYZ) accelerometer es un dispositivo capacitivo microtrabajado que contiene un filtro pasa-baja unipolar, acondicionamiento de señal inherente, compensación de temperatura y selección de sensibilidad (1.5/2/4/6g). El acelerómetro es ideal para sistemas de bajo consumo, cuando el consumo está alrededor de 500 µA cambia a modo *sleep*[8] donde el consumo es de 3 µA. El dispositivo también tiene la posibilidad de despertar del modo *sleep* de forma rápida. Este modo de funcionamiento tiene la ventaja de ser robusto y es ideal para altos usos de sensibilidad (800mV/g para 1.5g).

Dentro de SHIMMER el acelerómetro está conectado a la CPU vía tres canales ADC.

3.3.2.2 LEDs

La placa base contiene cuatro LEDs[9] (verde, amarillo, naranja y rojo) que pueden ser usados por el programador como indicadores de estado. Están conectados a la CPU vía GPIO (General Purpose Input/Output).

3.3.2.3 Botón de Reset

La placa base contiene un botón de reset que permite reiniciar la CPU. El botón está conectado via BSL[10].

3.3.2.4 Conexiones de Expansión

Para dotar al dispositivo de una funcionalidad adicional, la placa base posee dos conexiones de expansión, una interna y otra externa.

[8] Modo sleep - modo en reposo.
[9] LED - Light-Emiting Diode.
[10] BSL – Bootstrap Loader

La conexión interna permite al usuario conectar las placas de ECG[11], *kinematics* (sensor de movimiento), EMG[12], GSR[13] y EEG[14] a la placa base dependiendo de la aplicación requerida.

La conexión externa permite al usuario conectar a la placa el puerto de programación o el multicargador.

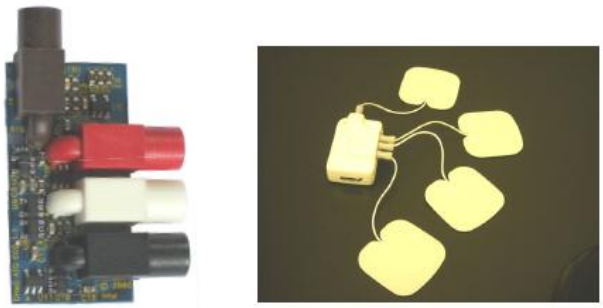

Figura 3.7: Extensión ECG y SHIMMER con sensores.

Figura 3.8: *Kinematics*

3.3.3 COMUNICACIONES

Una de las funciones claves de SHIMMER es su capacidad de comunicarse como una plataforma inalámbrica. SHIMMER tiene dos módulos de radio: BlueTooth y 802.15.4

3.3.3.1 Radio 802.15.4

SHIMMER contiene un transmisor de radio Chipcon CC2420 y la antena gigaAnt 2.4GHz Rufa.

[11] ECG - Electrocardiograma.
[12] EMG - Electromiograma.
[13] GSR - Galvanic Skin Response.
[14] EEG - Electroencefalograma.

El CC2420 es ideal para SHIMMER por como esta diseñado debido al bajo consumo de potencia y corriente (17.4mA para la transmisión y 18.8 mA para la recepción). La radio también puede ser apagada por el MSP430 para operaciones de bajo consumo de potencia.

La radio CC2420 tiene un módem de espectro expandido de secuencia directa (DSSS) y una tasa teórica de de transmisión de datos de 250Kbps.

La radio CC2420 está controlada por una conexión SPI[15] sobre el puerto USART1 y tiene soporte tanto para aplicaciones como para el manejo de paquetes, transmisión de datos, cifrado de datos, intensidad de la señal recibida, calidad del enlace y *frecuencia de transmisión*.

La radio CC2420 requiere sólo unos componentes adicionales como un oscilador de cristal de referencia y no necesita y filtros externos.

3.3.3.2 Radio *Bluetooth*

SHIMMER usa la clase 2 *BlueTooth* Roving NetworksTM RN-46 [8] para comunicarse mediante una antena integrada de 2.4Ghz.

El módulo Bluetooth está conectado al MSP430 por el puerto serie USART1. También puede ser controlado a través de cadenas ASCII sobre el enlace de radio frecuencia *Bluetooth*.

El módulo es ideal para operaciones de bajo consumo de potencia requeridas por SHIMMER, tiene cuatro modos distintos de consumo: *transmit* a 60mA, *reception* a 40 mA, *idle state*[16] a 1,4 mA, *deep sleep*[9] *a 50µa.*

El dispositivo RN-46 tiene un alcance de más de 10 metros y el consumo de potencia puede ser ajustado dependiendo de la distancia. El sistema tiene setenta y nueve canales en intervalos de 1MHz y ofrece una conexión segura y robusta mediante espectro ensanchado por salto en frecuencia (FHSS[10]) y esquemas de corrección de errores.

La conexión *Bluetooth* tiene una tasa de transmisión en baudios de 921.6Kbps sobre el USART1 y una free *space transmission rate* de 721Kbps, con una sensibilidad del receptor de -82dBm.

[15] SPI - *Serial Perifpheral Interfac.*

[16] Idle state – Estado en reposo

3.3.4 Almacenamiento de memoria

SHIMMER dispone de una ranura para una tarjeta MicroSD idónea para incorporar recursos extras de memoria.

3.3.5 Silicon ID Hardware

SHIMMER tiene un identificador DS2411 de silicio con un número de registro de 64 bits que contiene un número de serie único de 48 bits. Este ID es utilizado para obtener la dirección MAC que se utilizara en la transmisión a través de la radio 802.15.4, por ejemplo.

4. SIMULADORES DE REDES DE SENSORES

En la actualidad las redes de sensores inalámbricas se han convertido en un campo de estudio que se encuentra en continuo desarrollo.

Con el objetivo de medir el impacto de cambios en parámetros configurables de la red, en cuanto al rendimiento y a la estimación del consumo de potencia, se hacen sumamente necesarios simuladores y modelos de arquitecturas de sensores.

Por ejemplo podemos modificar la arquitectura de una WSN [17]antes de construirla en la plataforma real. Si queremos probar un nuevo protocolo MAC nosotros podemos implementarlo, depurarlo y probarlo en el simulador para obtener valores de consumo de potencia y de funcionamiento antes de implementarlo en el nodo real.

Esto ha propiciado la aparición de multitud de herramientas de simulación a fin de facilitar su estudio. Como ejemplos de simuladores de redes de sensores podemos encontrar los siguientes.

- TOSSim[9]
- NS-2[10]
- OMNeT++[11]

4.1 TOSSim

Es un simulador de eventos discretos para redes de sensores basadas en el sistema operativo TinyOS.

Ofrece la posibilidad de compilar las aplicaciones en TinyOS en el propio *Framework* de TOSSim en lugar de tener que compilar la aplicación para cada nodo. Esto permite a los usuarios depurar, testear y analizar algoritmos en un entorno controlable.

Características:

- **Fidelidad:** Captura el comportamiento de TinyOS a muy bajo nivel. Simula la red a nivel de bit y todas las interrupciones del sistema.

- **Time:** Desde la perspectiva de TOSSIM una porción de código se ejecuta de manera instantánea. Mientras que la velocidad del reloj es un valor

[17] WSN – Wireless Sensor Network

entorno a 4MHz. Esto provoca que algunos eventos no ocurran hasta que el código se complete en lugar de ocurrir en el preciso instante.

- **Modelos:** No modela el mundo real, pero permite abstracciones de este. Mediante herramientas los usuarios pueden implementar los modelos que deseen.

- **Radio:** TOSSim no modela la propagación de la radio, si no que provee una abstracción de la radio directa e independiente con bits de errores entre dos nodos. Mediante programas externos se puede recrear el modelo de radio deseado y mapearlo en los bits de errores mencionados anteriormente.

- **Potencia:** Tampoco modela el consumo de energía, pero mediante anotaciones en cada componente es sencillo calcular el consumo de energía.

- **Construcción:** Para simular un sistema o un protocolo, se debe escribir una implementación en TinyOS de éste. Por un lado es más difícil abstraer la simulación pero por otro puedes ejecutar tus aplicaciones en los nodos actuales.

- **Redes:** TOSSim simula redes mica RFM con una pila de 40KB, incluyendo la capa MAC, encriptación y confirmaciones.

- **Imperfecciones:** El hecho de que TOSSIM capture el funcionamiento a bajo nivel puede provocar que programas que no funcionen en la vida real si que funcionen en las simulaciones. Por ejemplo una interrupción muy larga puede provocar que el nodo real se rompa mientras que en el simulador funcionará sin ningún problema.

En la siguiente figura podemos ver una imagen del simulador.

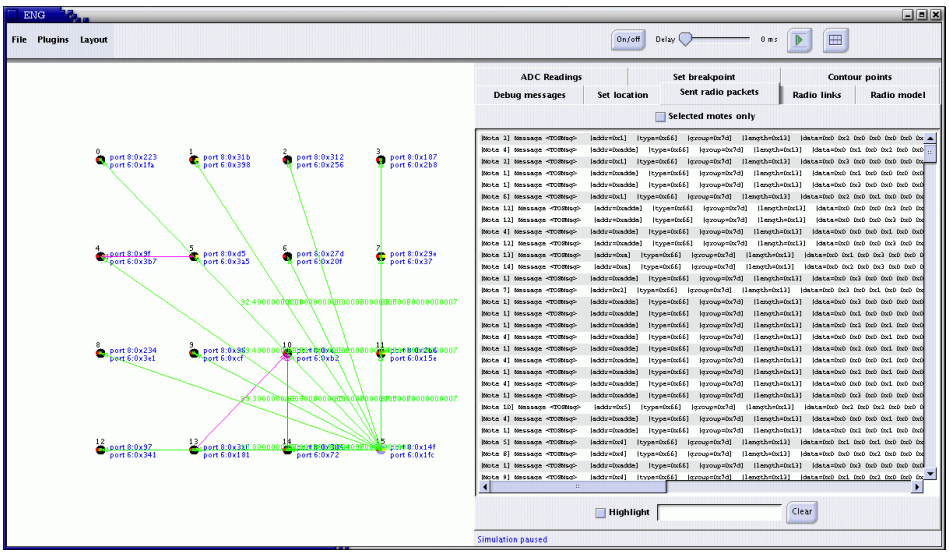

Figura 4.1: Simulador TOSSim.

4.2 NS-2

Simulador de eventos discretos dirigido a la creación de redes de investigación. Nació en el año 1989 como una variante del simulador de redes REAL [13]. Proporciona un apoyo sustancial a la hora de simular redes de cable o inalámbricas, simulando el protocolo TCP, enrutamiento y protocolos *multicast*[18]. Actualmente la última versión es la 2.34 de Junio de 2009.

En la siguiente figura podemos ver la interfaz de NS-2

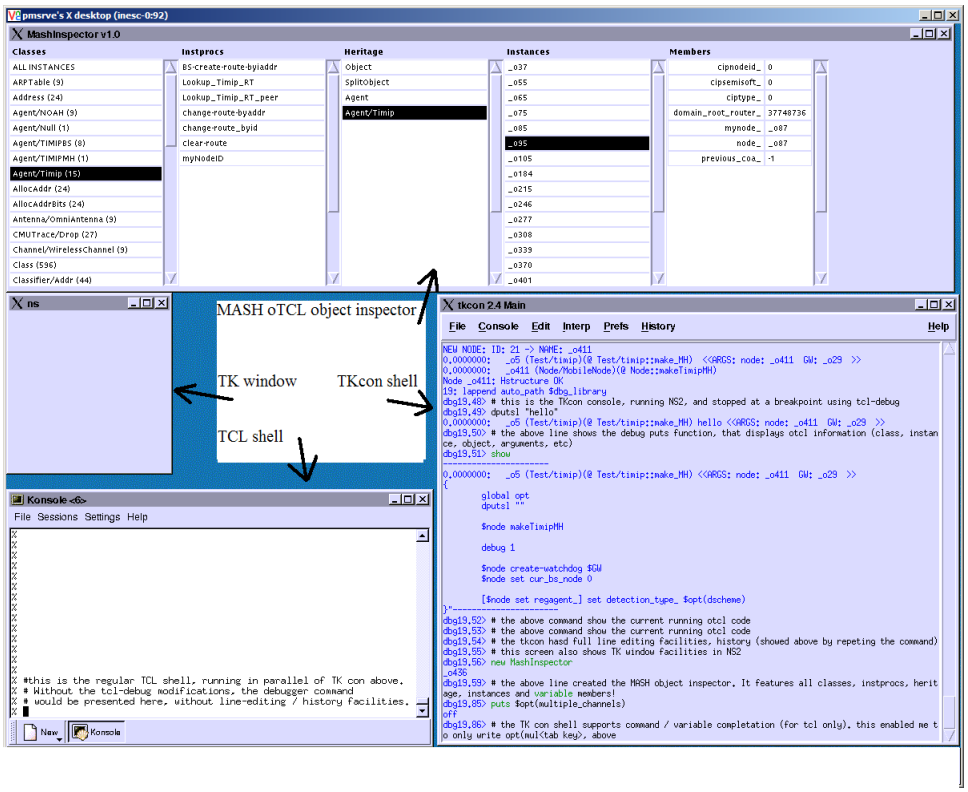

Figura 4.2: Simulador NS-2.

4.3 OMNeT++

Entorno de simulación de eventos discretos. Su primera área de aplicación fueron las redes de sensores, pero debido a su arquitectura flexible es posible usarlo en otras áreas de simulación. Además proporciona una arquitectura de componentes para los modelos. Los componentes están programados en C++ y luego ensamblados en grandes componentes usando lenguajes de alto nivel. Cuenta con una amplia

[18] Multicast – Termino empleado en redes para referirse a la multidifusión.

interfaz de usuario, y debido a su arquitectura modular, la simulación del núcleo y modelos pueden ser incorporadas fácilmente en sus aplicaciones.

En la siguiente figura podemos ver la interfaz de OMNeT++

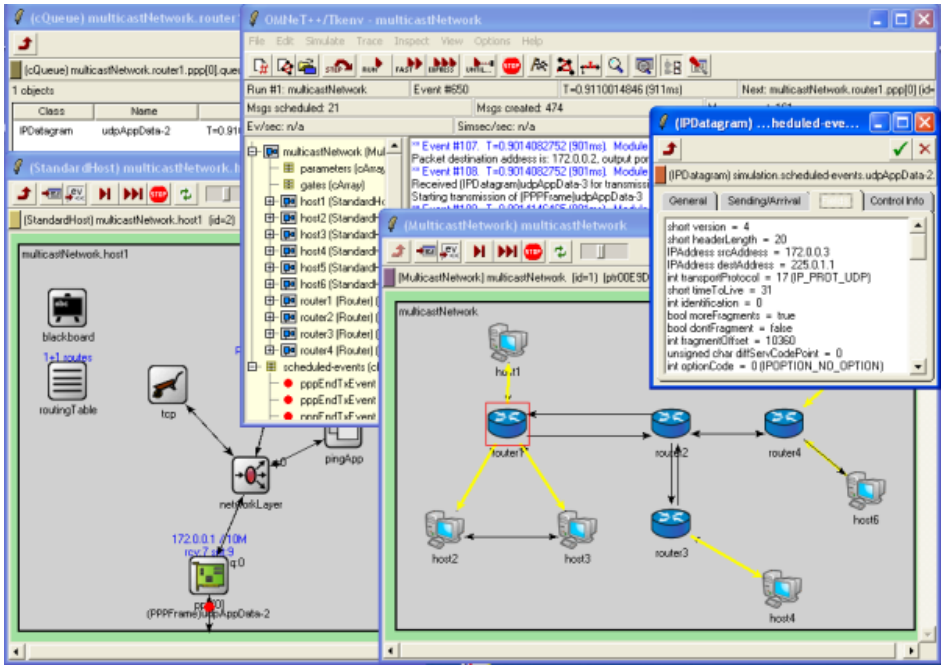

Figura 4.3: Simulador OMNeT++.

En la siguiente tabla podemos encontrar los principales simuladores de RDS y las principales características soportadas por ellos.

Simulation Tool	TOSSIM	ns-2	GloMoSim (Qualnet)	OMNeT++ (Omnest)	OPNET	Avrora	EmStar/EmTOS	J-Sim (JavaSim)	COOJA	JiST / SWANS	SwarmNet/Shawn
Simulator/Emulator	Simulator, Emulator(AVR, MSP)	Simulator	Simulator	Simulator	Simulator	Simulator/ Emulator(AVR)			Simulator, Emulator (MSP)		Simulato
Status	Supported	Supported	GlomoSim: Not supported, Qualnet: Supported	Supported	Supported	Supported but not being actively developed	Inactive (Last update 2005)		Actively developed		Supported
language of the tool	Java	C++, OTcl	Parsec (C, C++)	C++, NED	C, C++	Java	Inactive (Last update 2005)	Java	Java	Java	C++
language of the models	nesC	C++, OTcl		C++, NED	C, C++	?	Inactive (Last update 2005)		?		C++
Standards supported	802.15.4	AODV, OLSR, DYMO, 802.11, Bluetooth, Mobile IP	CSMA, IEEE 802.11, MACA, IP with AODV, Bellman-Ford, DSR, Fisheye, LAR scheme 1, ODMRP, WRP, TCP, UDP, CBR, FTP, HTTP, Telnet + more	802.11	802.11, 802.16, MANET, MobileIP						Simulato
Supports Energy Consumption Research	with extension PowerTOSSIM										

Figura 4.5: Comparativa de simuladores.

5. Simulador MSPSim

MSPSim es un simulador a nivel de instrucción para el microcontrolador MSP430, que está diseñado para ser usado como componente en una red de sensores más grande que soporte simulación *cross-level*[19]. Por esta razón MSPsim está preparado para controlar múltiples instancias del simulador en un sólo proceso a diferencia de otros simuladores de MSP430 tales como GDB MSP430.

MSPsim también contiene un simulador de placas que se utiliza para modelar el comportamiento hardware de periféricos tales como puertos de comunicación, LEDs y dispositivos de sonidos como por ejemplo un *beeper*[20]. El diseño del MSPSim junto con su implementación en Java facilita adaptar el simulador a nuevas placas de sensores.

El simulador se puede ampliar fácilmente con periféricos que hagan posible simular varios tipos de nodos basados en el microcontrolador MSP430

5.1. Descripción de MSPSim

MSPSim es un simulador basado en java utilizado para simular el comportamiento de programas en arquitecturas compuestas por microprocesadores MSP430. En concreto permite la simulación de nodos TMoteSky [15] y ESB [16].

- Nodo TMoteSky:

  Las principales características de este nodo son:

  - TI MSP430F1611 microcontrolador hasta 8 MHz .
  - 10k SRAM, 48k Flash + 1024k almacenamiento serie.
  - 250kbps 2.4 GHz Chipcon CC2420 IEEE 802.15.4 Wireless Transceiver
  - Sensores de humedad, temperatura y luz en placa.
  - Consumo de energía mínimo.
  - Programación vía USB
  - Serial ID chip

[19] Cross-level – Distintos niveles.
[20] Beeper – Dispositivo de sonido.

- Nodo ESB:

  Entre sus características cabe destacar las siguientes:

  - Microcontrolador de Texas Instruments MSP430
  - 2KBytes de ram y 60KBytes de flash rom
  - Radio TR1001
  - Una EEPROM de 32 KBytes

En la figura 5.1 podemos observar el sistema de simulación completo del MSPsim con un microprocesador MSP430 y periféricos conectados.

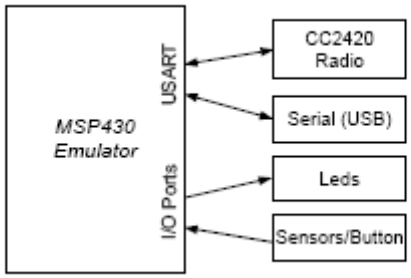

Figura 5.1: Esquema de MSPSim

MSPSim soporta la carga de firmware en formato *IHEX*[21] y *ELF*[22], *y* posee las siguientes herramientas que permiten la visualización y control de la simulación en tiempo de ejecución, dando control total de la misma.

- Monitor de pila.
- Ciclo de trabajo.
- USART port output.
- Panel de control de la simulación.
- Imagen física del nodo.
- Consola MSPSim para control de la simulación.

[21] IHEX – Formato de código compilado de Intel.
[22] ELF – Formato de código compilado.

Monitor de pila: Muestra el **maxstack** y el **stack** de la pila.

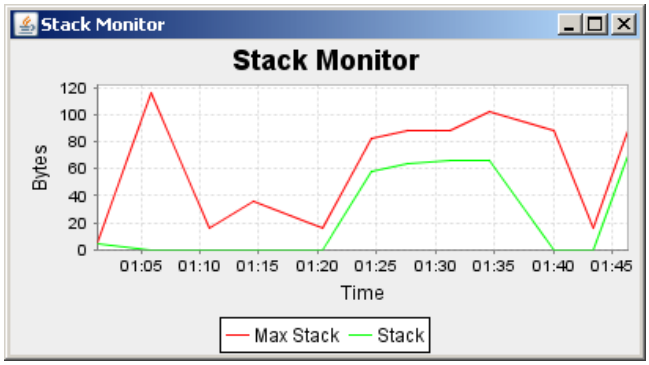

Figura 5.2: Monitor de pila.

Ciclo de trabajo: Es el monitor de uso de cada componente del nodo. Mide el uso de la CPU en porcentaje, el trabajo de la radio (transmitiendo y recibiendo) y la activación y desactivación de los LED.

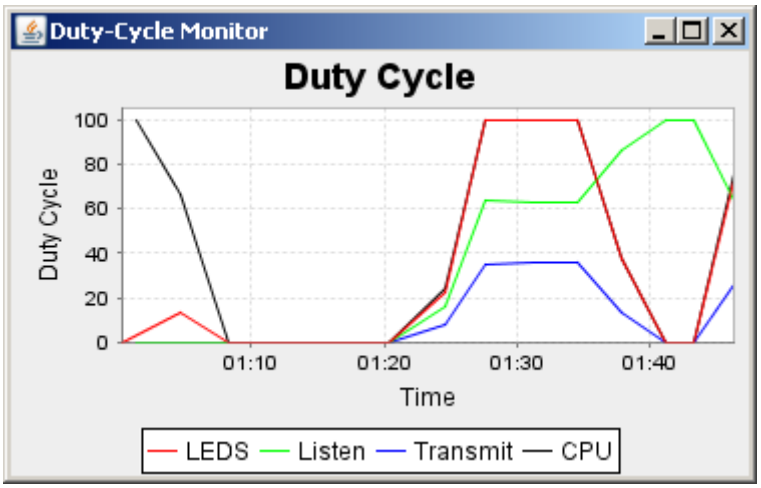

Figura 5.3: Monitor de ciclo de reloj.

USART port output: Utiliza el dispositivo periférico puerto serie para transmitir/recibir de manera síncrona.

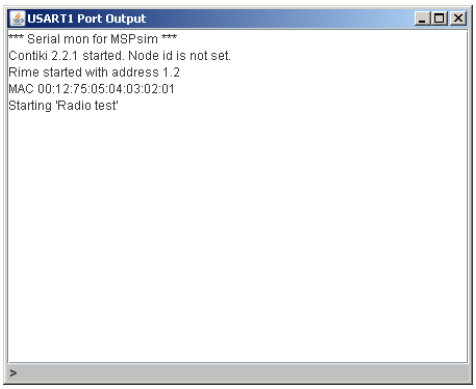

Figura 5.4: USART1 port output.

Panel de control de la simulación: Es probablemente la herramienta mas importante del simulador. Entre sus funciones se encuentran:

- **Visualización del código:** Ventana que muestra el código maquina en tiempo de ejecución.
- **Debug On:** Capacidad para activar o desactivar el depurador. Este depurador muestra los resultados en la consola MSPSim
- **Single step:** Posibilidad de ejecución en modo *step by step* (paso a paso).
- **Run/Stop:** Para/arranca la ejecución del programa.
- **Stack trace**: Visualizar la pila de ejecución.
- **Profile dump:** Muestra estadísticas de uso del sistema. Ciclos en ejecución, la media de ciclos de por cada vez que se ejecutó y el numero de llamadas al procedimiento y su media.
- **Show Source:** Muestra el código fuente del programa que se esta ejecutando.

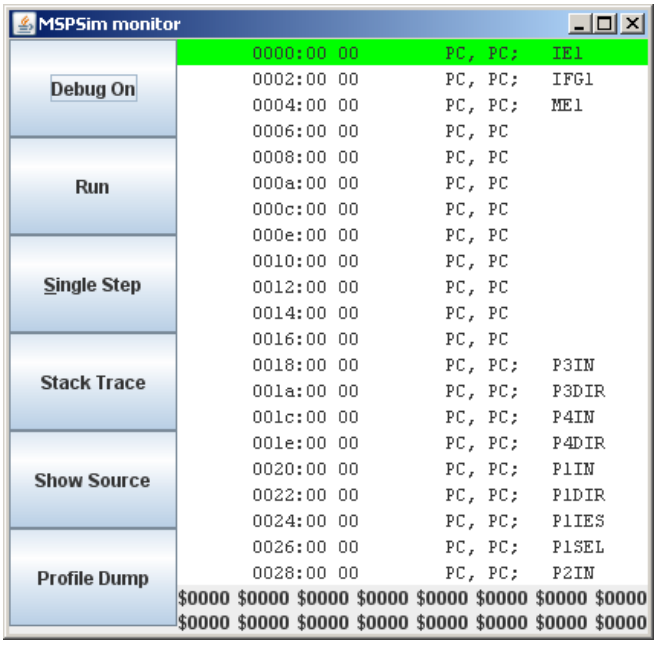

Figura 5.5: Monitor general de MSPSim.

Imagen física del nodo: Se puede ver como se encienden y apagan los LEDs y en el caso particular del ESB al pulsar el botón que aparece en la imagen, permite lanzar el evento que ese botón active.

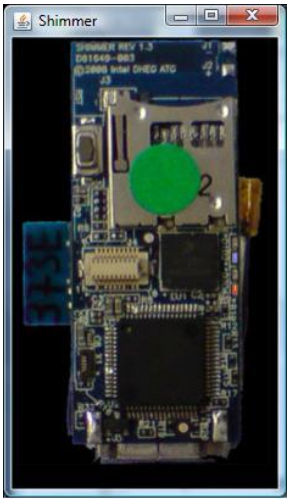

Figura 5.6: Imagen del nodo.

Consola MSPsim: Es la consola del entorno de trabajo. En esta consola se permite realizar cierto tipo de tareas entre las que cabe destacar:

- Monitorizar la radio.
- Imprimir el valor de direcciones de memoria.
- Ver el log generado por las llamadas y los eventos.
- Resetear la CPU, arrancarla etc.
- Dar valores concretos a posiciones de memoria en tiempo de ejecución.

En definitiva controlar y monitorizar cierto tipo de operaciones sobre el nodo y la aplicación que está ejecutándose en él.

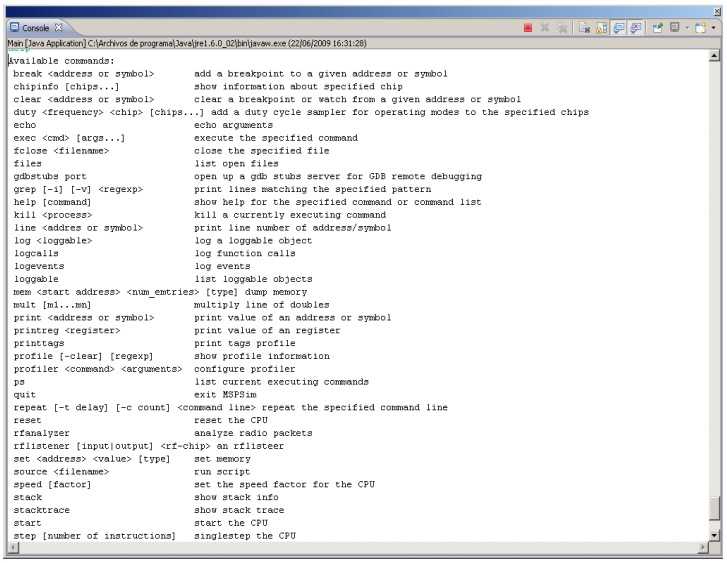

Figura 5.7: Consola MSPSim.

En la figura 5.7 podemos observar un extracto de los posibles comandos que admite la consola de MSPSim.

5.2 Nuestra labor con MSPSim

Objetivos de nuestro trabajo:

- Añadir un gráfico con la evolución del consumo de energía en el tiempo.
- Capturar el tráfico de la radio generado por el nodo e interpretar cada paquete.
- Ficheros de Log donde poder estudiar la información recabada por el simulador sobre el consumo de potencia y el trafico generado por la radio.
- Modificar las referencias en el código al tipo de nodo TMoteSky para asociarlo de ahora en adelante como nodos del tipo Shimmer. Aparte se le han añadido características típicas del SHIMMER.
    - Un LED naranja, pasa de tener 3 LEDs a tener 4 LEDs.
    - Modificar los pines mediante los cuales se conectan los diferentes componentes del nodo.
    - Parametrizar el tamaño de la memoria del MSP430 y de la radio CC2420.

En el resumen anterior se comentan las diferentes modificaciones que ha sufrido el original MSPSim para adaptarlo a nuestras necesidades. Estas adaptaciones no han resultado triviales, cualquier modificación de código existente implica un cierto nivel de comprensión de la arquitectura de la aplicación que solo se consigue investigando en profundidad la herramienta.

Una vez adquirido cierto conocimiento sobre el simulador y sus características, diseño de clases y paquetes, nos fue relativamente sencillo realizar las modificaciones deseadas.

5.2.1 Calcular la energía consumida por el sistema

Hay dos fuentes principales de consumo de energía en el nodo, el microcontrolador y la radio. Típicamente la radio consume del 50 % al 90 % del consumo total de energía.

Así, un modelo de energía de la radio es obligatorio para un análisis eficaz de la plataforma entera. Para algunos usos sin embargo, el consumo de energia del microcontrolador podría no ser insignificante. Así decidimos incluirlo en nuestro modelo.

El primer objetivo era conseguir algo como lo que nos muestra la figura 5.8 un grafico que en intervalos definidos calcule el consumo de potencia en ese instante. Para modificar consumos de los distintos componentes nos hemos

ayudado de un fichero de configuración externo, al que se le dan los valores deseados para cada elemento.

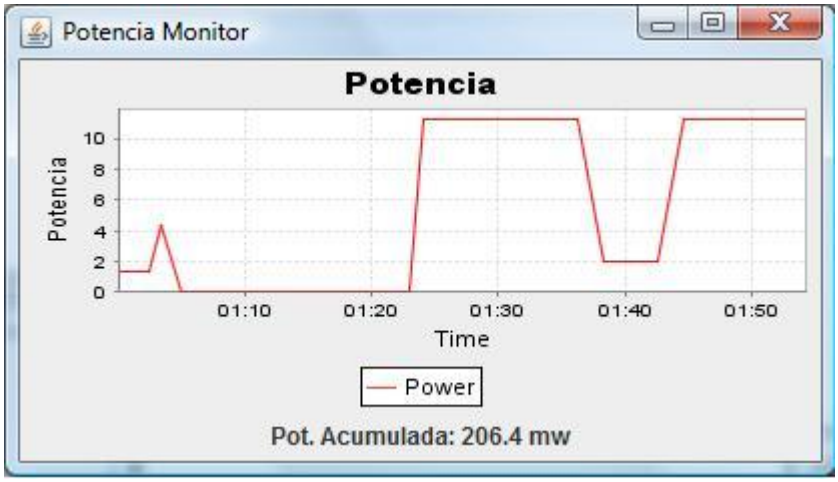

Figura 5.8: Monitor de consumo de energia.

El cálculo de la potencia se realiza por muestreo a partir de la siguiente información:

- **Consumo de la CPU:** El consumo en potencia de la CPU se mide en función de su porcentaje de uso, si la CPU está funcionando a más del 20% consideramos que el consumo en potencia es superior y por eso se le da el valor definido en el atributo *potHICPU* del fichero de configuración. Por el contrario, si el porcentaje de uso es mejor del 20%, entonces se considera que el consumo de potencia es inferior y se utiliza el valor del atributo *potLOWCPU*.

  Como valores de medida hemos tomado, que si el voltaje es de 2,8 V, el microcontrolador consume 2 mA de corriente cuando el porcentaje de uso es superior al 20 % y de 0,66 mA en caso contrario.

- **Consumo de la Radio:** El consumo de energía por parte de la radio se basa en el estado de la misma, es decir, si esta transmitiendo o recibiendo. Si esta recibiendo se utiliza el valor del atributo *potListen*, por el contrario si la radio esta transmitiendo se utiliza el valor del atributo *potTrans*.

  Los valores están tomados de valores experimentales. Cuando el voltaje es de 2,8 V el consumo de energia de la radio en modo transmitiendo es de 24,82 mA y en modo recibiendo es de 17,54 mA.

- **Consumo de los LEDs:** Si se encuentra algún Led activado se utiliza el valor de *potLeds* para modelar su consumo de energía.

La siguiente figura muestra los consumos de energía utilizados para la simulación, estos valores se han obtenido de un procesador y radio de características similares.

	LED	Radio		CPU	
		Recibiendo	Transmitiendo	Modo ahorro	Alto rendimiento
Consumo	2	69.496	49.112	1.32	5.6

Figura 5.1: Consumos utilizados.

Una vez definido el consumo de energía de cada componente, se suman y se muestran en la grafica correspondiente. El muestreo se realiza cada 100 milisegundos lo que nos permite obtener unos valores adecuados sin llegar a sobrecargar el sistema muestreando información de más.

En este grafico solo se puede mostrar el consumo de potencia en determinados instantes de tiempo lo que no es del todo útil, o al menos, debería complementarse con la evolución de la potencia en el tiempo, es decir, cual es la potencia acumulada en cada instante de tiempo. Con esta nueva métrica podemos medir el consumo global de una aplicación en el nodo, lo que resulta realmente interesante a la hora de calcular cuando puede durar la batería del nodo ejecutando cierto programa.

5.2.2 Monitorizar el tráfico de la radio

En las redes de sensores es muy importante conocer en todo momento lo que envía y recibe cada nodo y de este modo conocer las tasas de paquetes enviados y recibidos. Tan importante es conocer la información mencionada anteriormente como interpretar cada paquete enviado y recibido por los nodos.

En MSPSim la radio funciona de la siguiente manera, cuando se detecta que se ha de enviar un paquete, se envía el preámbulo por parejas de bytes y a continuación se procede a enviar el resto del paquete por parejas de bytes. Lo que hacemos es capturar todos los pares de bytes enviados que hay entre un preámbulo y el siguiente, así sabemos cuando acaba cada paquete. Es el mismo método que emplea cualquier tarjeta de red actual.

Una vez capturada el paquete entero se procede a su interpretación campo por campo. Esta interpretación se ha hecho en base al estándar de la radio 802.15.4 que se explica en el anexo A.

5.2.3 Generación de ficheros log

No solo es interesante obtener resultados en tiempo de ejecución, también es muy importante poder guardar un historial de todos los datos a modo de base de datos de pruebas a fin de poder comparar los resultados obtenidos introduciendo modificaciones en el programa que ejecute el simulador.

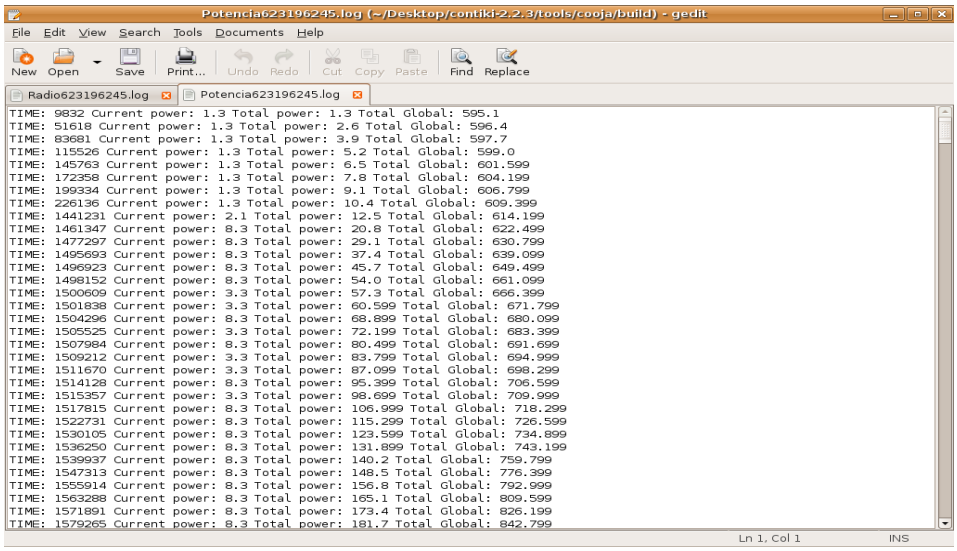

Figura 5.1: Log de la potencia.

En la figura 5.1 Se muestra el formato que tiene un fichero de log de energía consumida. Cada 100 milisegundos se muestran los 3 valores que interesan al estudio, el consumo en ese momento, el consumo acumulado y el consumo total. Este último valor puede verse modificado al introducir mas nodos en la simulación, que de hecho es lo que se hará con la herramienta COOJA.

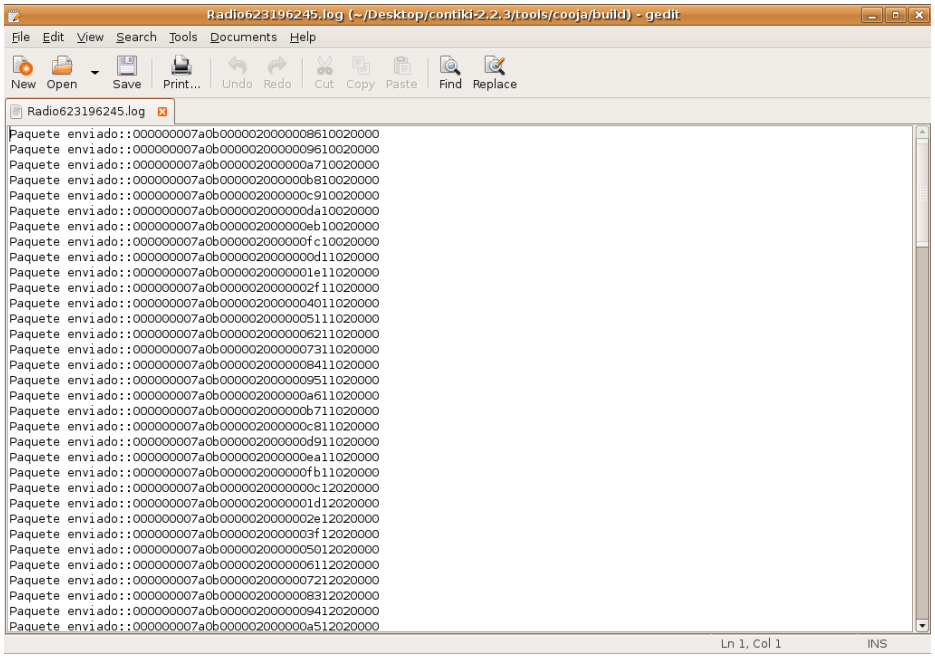

Figura 5.2: Log de la radio.

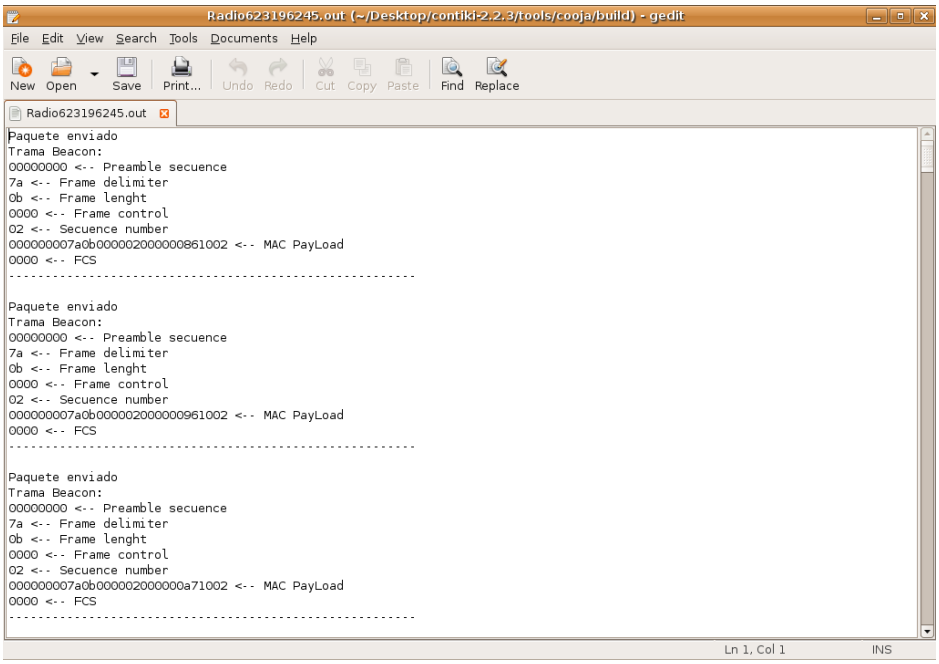

Figura 5.3: Log de interpretación de paquetes.

Las figuras 5.2 y 5.3 muestran en el primer caso el paquete enviado/recibido por el sensor en formato hexadecimal. En el segundo caso, se puede observar la interpretación para cada paquete siguiendo el estándar.

5.2.4 Adaptaciones de TMoteSky para adecuarlo al SHIMMER

Pese a ser realmente similares los nodos TMoteSky y SHIMMER (ambos utilizan procesadores de la familia MSP y radio CC2420) tienen algunas diferencias de las cuales nos interesa incorporar al simulador; el cuarto LED que posee el SHIMMER, modificar las conexiones internas entre los componentes, los pines y una modificación para realizar experimentos, como es modificar los tamaños de las memorias del procesador y de la radio.

Estas modificaciones significan cambiar ciertos valores en las clases del simulador para realizar las nuevas asociaciones de pines. Incluir el nuevo LED requiere incluir ciertos métodos de más para realizar las comprobaciones pertinentes y comprobar su estado (encendido/apagado).

Llegados a este punto tenemos una herramienta bastante potente y veraz que ayudará en el proceso de simulación de aplicaciones en la plataforma SHIMMER evitando costes innecesarios y permitiendo trabajar de forma más ágil que programando el nodo cada vez que se deseara realizar una prueba.

6. Simulador COOJA

Los sensores están ahora integrados en todo tipo de dispositivos tales como PDAs y teléfonos. Estos dispositivos están conectados a móviles, robots, vehículos o animales. La colección de datos de estos sensores móviles representa desafíos relacionados con la viabilidad de la topología de la red de sensores y la necesidad de limitar la comunicación para conseguir consumos de energía menores y optimizar los anchos de banda de comunicaciones.

De ahí que simuladores de esta índole sean imprescindibles para poder estudiar el sistema de una forma muy pareja a la realidad y así optimizarlo para el caso real.

COOJA [17] es un simulador basado en java, diseñado para simular redes de sensores que se ejecutan bajo el sistema operativo Contiki [18].

6.1 Descripción

COOJA simula nodos de RDS[23] donde cada nodo puede ser de diferente tipo, no solo en el software del nodo sino también en el hardware de la simulación. COOJA es flexible en muchos de sus aspectos, se puede extender fácilmente su funcionalidad introduciendo modificaciones.

Un nodo simulado en COOJA tiene tres propiedades básicas:

- Memoria de datos
- Tipo de noto
- Hardware periférico

El tipo del nodo se puede compartir entre varios nodos de la simulación y determina las propiedades comunes a todos estos nodos. Por ejemplo, los nodos del mismo tipo, tienen el mismo código de programa simulado en el mismo hardware periférico.

COOJA actualmente es capaz de ejecutar programas en Contiki de dos formas diferentes. O bien ejecutando el código del programa como código nativo directamente en la CPU, o bien ejecutando el código del programa compilado a nivel de instrucción en el emulador de MSP430. También es capaz de simular nodos que no estén bajo sistema operativo Contiki, como nodos implementados en java o incluso nodos que ejecuten otro sistema operativo. Cualquiera de los distintos enfoques tiene ventajas e inconvenientes. Los nodos basados en java permiten simulaciones más rápidas pero no ejecutan el código desplegable. Los nodos emulados proporcionan un nivel de detalle de

[23] RDS – Redes de Sensores

la ejecución mayor que los que proporcionan los nodos basados en java o los nodos en código nativo. Por ultimo los nodos en código nativo permiten simulaciones mas rápidas que los nodos emulados pero si que ejecutan el código desplegable.

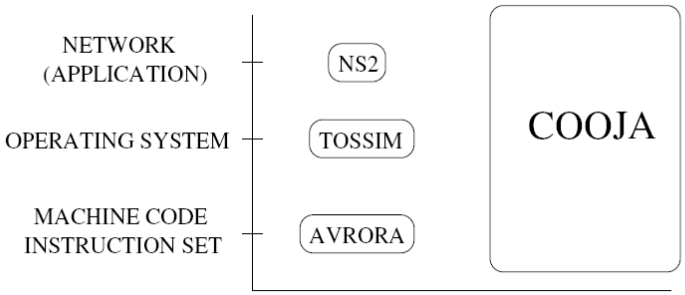

Figura 6.1: COOJA puede simular a varios niveles

6.1.1 Simulación *cross-level*

Como se muestra en la figura 6.1 COOJA permite simultáneamente simulación a tres niveles: a nivel de red (o aplicación), a nivel de sistema operativo y a nivel de código de instrucción.

Un nodo individual siempre es simulado a uno de estos niveles, sin embargo la principal ventaja que proporciona la simulación *cross-level* es que nodos a diferentes niveles de simulación pueden coexistir en la misma simulación.

6.1.2 Modelos de radio.

Cualquier simulación en COOJA utiliza un modelo que caracteriza la propagación de las ondas de radio. Nuevos modelos de radio pueden ser añadidos al entorno de simulación. El modelo de radio a utilizar en la simulación se elige cuando comienza la simulación.

Los modelos de radios permitidos por COOJA son:

- UDGM [24][19]
- DGRM

[24] UDGM – Unit Disc Graph Medium

6.1.3 Interfaz de usuario

Uno de los objetivos de COOJA es la extensibilidad. Mediante interfaces y plugins lo podemos extender de manera sencilla. Una interfaz representa un sensor de una propiedad de un nodo, o un dispositivo como un a posición, un botón o un transmisor de radio. Un plugin se utiliza para interactuar con una simulación, por ejemplo para controlar la velocidad de simulación o visualizar todo el tráfico de la red entre los nodos simulados.

6.1.4 Ejemplo de utilización

A continuación vamos a detallar los pasos a seguir para simular una aplicación. La aplicación es bastante sencilla: cuando iniciamos la simulación los nodos envían por *multicast* tramas de conexión a todo el resto de nodos.

Comenzar la simulación:

Cuando comienza la simulación, utilizando el botón start de COOJA, nuestros nodos comienzan a transmitir. Durante la ejecución podemos interactuar con la aplicación: cambiando la velocidad de la simulación, mostrando diferentes características de nuestros nodos, configurando los valores para tramas perdidas, cambiar el rango de transmisión etc. como pueden ser el alcance de las transmisiones, el estado de los nodos.

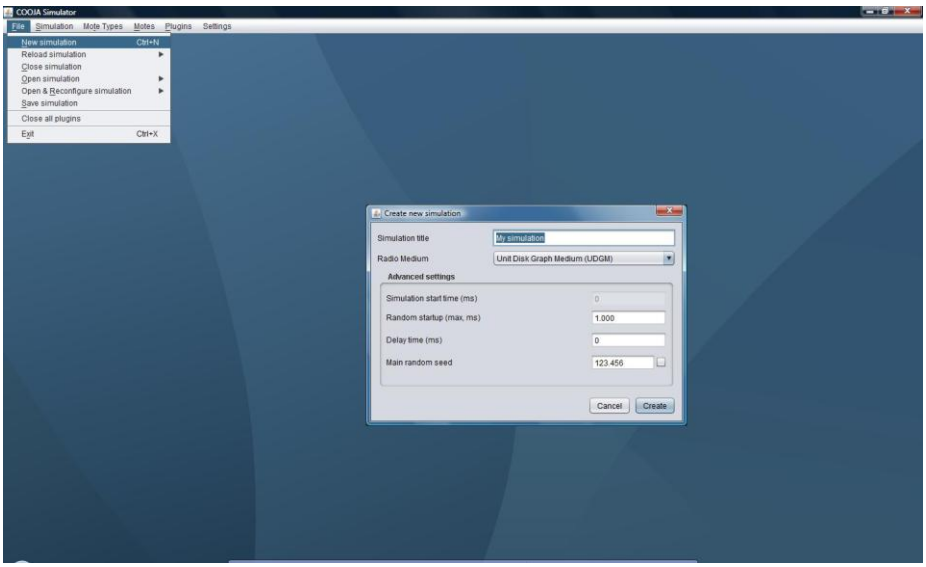

Figura 6.1: Crear simulación.

Creación del tipo de nodo:

Creamos la simulación usando el estándar de radio UDGM y el tipo de nodo seleccionando los archivos de código fuente como podemos observar en la siguiente figura.

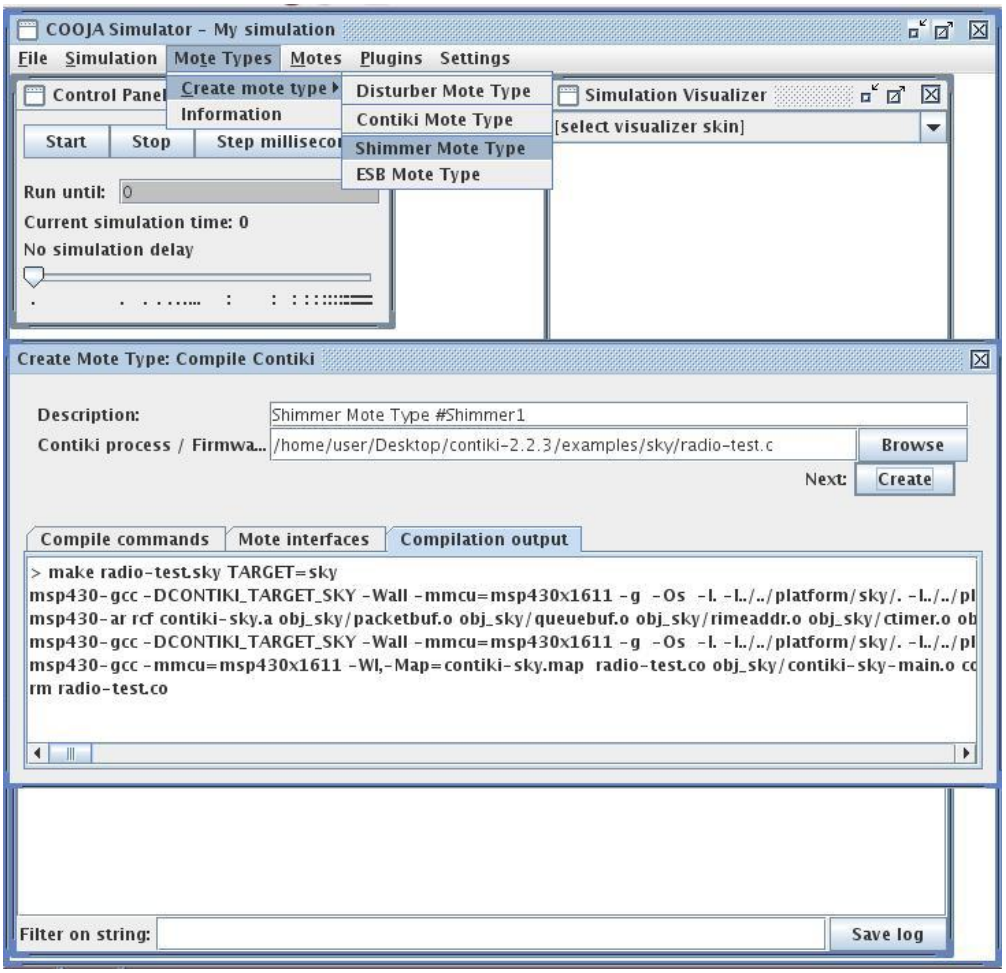

Figura 6.3: Creación de nodos.

Añadir nodos

Una vez que hemos terminado de compilar nuestros tipos de nodos ya estamos en disposición de añadir los nodos al dibujo de la red. En la siguiente figura mostramos las opciones de que disponemos.

Números de nodos: Aquí seleccionamos el número de nodos que queremos añadir a la simulación del tipo que estamos creando.

Dirección IP: Elegimos las direcciones IP de nuestros nodos.

Distribución de los nodos en el esquema de la red: Aquí podemos elegir la forma en la que se van a colocar los nodos en la simulación. Las posibilidades van

desde que los nodos se coloquen de manera aleatoria, hasta que se coloque de forma distribuida, en forma de círculo, elipse, malla y diversas figuras.

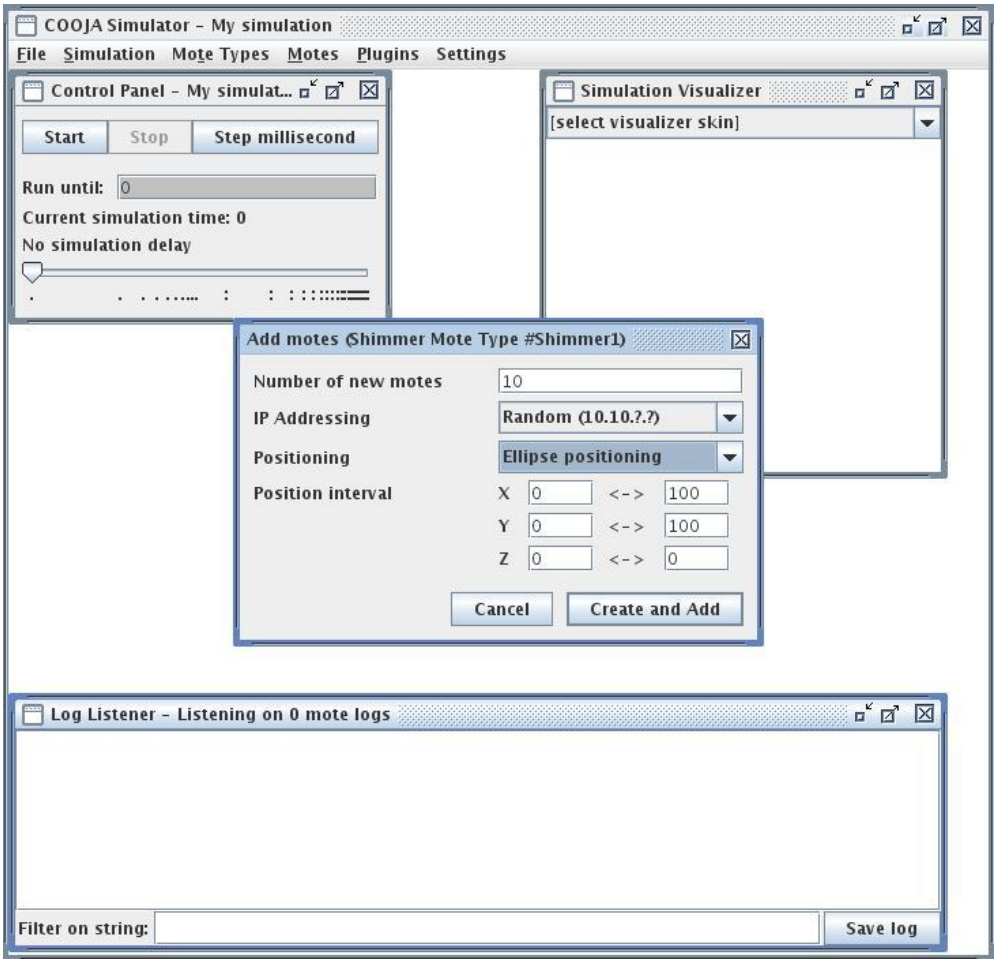

Figura 6.4: Añadir nodos a la simulación.

Cambio de estado de los LEDS de nuestros nodos

En esta aplicación los LEDs se encienden y se apagan en función del trabajo que este realizando el nodo. En el caso de que el LED que este encendido sea el rojo significará que el nodo esta enviando un paquete de radio, si el LED que se encuentra encendido es el verde el nodo estará recibiendo un paquete de radio, en el caso de que el nodo este al mismo tiempo enviando y recibiendo el LED que se encontrará encendido será el azul.

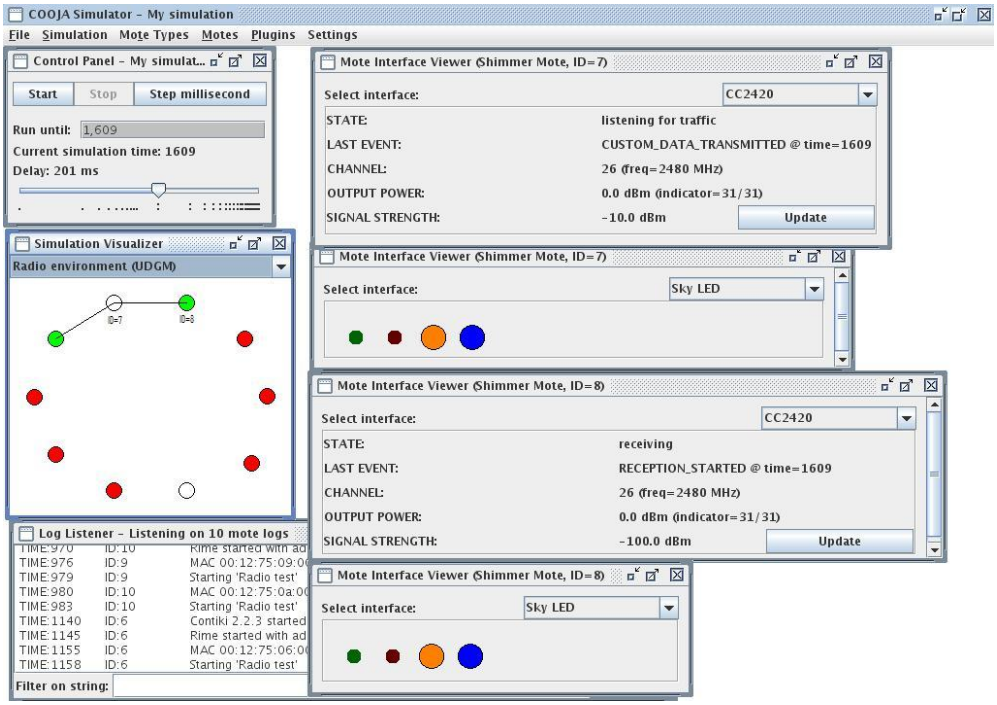

Figura 6.5: Ejemplo de simulación en COOJA

6.2 Nuestra labor con COOJA

Con todo esto, las únicas modificaciones necesarias en COOJA han sido la incorporación del cuarto LED que trae la arquitectura SHIMMER y la adecuación de las numerosas clases, para sustituir las referencias a nodos de tipo TMoteSky por SHIMMER.

Ha sido necesario modificar los plugins así como las clases de interconexión entre COOJA y MSPSim. Con los actuales entornos de desarrollo con los que contamos, como Eclipse [20] o NetBeans [21], no ha supuesto un gran esfuerzo realizar esta tarea, resumiéndose en renombrar clases y métodos.

7. Caso de estudio

Como caso de estudio hemos utilizado una aplicación incluida en la plataforma COOJA, esta aplicación establece una conexión entre dos nodos y se envían información mutuamente. Se ha procedido a ejecutar dicha aplicación primero en un nodo solo utilizando el simulador MSPSim obteniendo los siguientes resultados.

	LED	RADIO	CPU	Total
Totales	374	18558400	692,12	18559466,1
Porcentajes	0,0020%	99,9957%	0,0037%	100%

Figura 7.1: Resultados en MSPSim.

Los valores están dados en mJ y desglosados por componente. Como se aprecia en la tabla, el mayor consumo de la aplicación viene dado por el uso de la radio, como se comento anteriormente la aplicación simplemente envía y recibe información sin procesarla de ninguna forma, por ello el consumo de CPU es tan bajo con respecto al de la radio.

El trabajo a realizar ahora es ejecutar la misma aplicación bajo la plataforma COOJA y obtener los resultados de consumos de potencia bajo ciertas condiciones en la radio. Vamos a simular un entorno perfecto en el que no se pierden paquetes y los nodos están al alcance unos de otros.

La siguiente figura se muestra los datos globales para dos nodos que ejecutan la misma aplicación.

	LED	RADIO	CPU	Total
Totales	1057,34946	34520926,3	370583,382	34892567
Porcentajes	0,0030303%	98,9349%	1,0620697%	100%

Figura 7.2: Resultados en COOJA.

Como muestra la figura los porcentajes de consumos para una red de nodos y un nodo aislado son bastante similares. De ahí la importancia de optimizar, sobretodo, el consumo de energía por parte de la radio.

8. TRABAJO FUTURO

El trabajo realizado nos facilita el camino para la puesta en marcha en el nodo SHIMMER de aplicaciones reales, pues bien, este es el siguiente paso a realizar. Ejecutar pruebas reales de aplicaciones ECG y calcular los consumos de potencia reales y comprarlos con los simulados, sacar porcentajes de paquetes enviados satisfactoriamente, esto es, paquetes enviados por el nodo y que fueron recibidos correctamente, el número de paquetes reenviados y demás información de utilidad.

Una vez recabada información suficiente se procederá a estudiar y analizarla con el fin de encontrar los parámetros que nos permitirán optimizar el sistema. Esta optimización nos permitirá realizar un uso más eficiente de la tecnología empleada.

En otra línea de trabajo, podría ampliarse el simulador MSPSim para modelar de una manera más fiel a la realidad la arquitectura de SHIMMER. Actualmente las características modelas en el simulador son las estrictamente necesarias para el uso de aplicaciones ECG, pero llegado el momento, se puede requerir funcionalidad del Shimmer no implementada en el simulador.

Las características a implementar serian:

- Radio BlueTooth, simular el comportamiento de la radio BlueTooth que trae la plataforma Shimmer y realizar un trabajo similar al realizado con la radio CC2420, es decir, capturar los paquetes enviados y recibidos así como interpretarlos.
- Shimmer también incorpora una ranura MicroSD que aun no ha sido incluida en el simulador.

Llegados a este punto, seria imprescindible poder simular en COOJA el trafico de la radio BlueTooth así como el de la memoria MicroSD aportando estadísticas de uso, números de accesos, tiempo que tarda en cada acceso a la ranura etc.

Bibliografía

Body sensor networks - Guang-Zhong Yang, Editorial Springer

[1] Redes Ad-hoc
http://en.wikipedia.org/wiki/Ad_hoc

[2] Multi-hop
http://users.ece.utexas.edu/~rheath/research/multihop/

[3] Estándar IEEE 802.15.4
http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf

[4] Estándar 802.11
http://www.ieee802.org/11/

[5] Estándar IEEE 802.15.1
http://www.ieee802.org/15/pub/TG1.html

[6] SHIMMER
http://www.trilcentre.org/technology_platform/hardware.568.460.html

[7]Microprocesador MSP430
http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?sectionId=95&tabId=1200&familyId=342&family=mcu

[8] *Blueetooth* RovingNetworksTM RN-46
http://www.rovingnetworks.com/

[9] TOSSim
http://www.eecs.berkeley.edu/~pal/research/tossim.html

[10] NS-2
http://www.isi.edu/nsnam/ns/

[11] OMNeT++
http://www.omnet-workshop.org/2009/

[12] OPENET
http://www.opnet.com/

[13] REAL
http://www.cs.cornell.edu/skeshav/real/overview.html

[14] RDS
http://www.rds.org.uk/

[15] TMoteSky
http://www.sentilla.com/moteiv-endoflife.html

[16] ESB
https://wiki.ti5.tu-harburg.de/wsn/scatterweb/esb

[17] COOJA
http://www.sics.se/~fros/cooja.php

[18] Contiki
http://www.sics.se/contiki/

[19] UDGM
http://en.wikipedia.org/wiki/Unit_disk_graph

[20] Eclipse
http://www.eclipse.org/

[21] NetBeans
http://www.netbeans.org/

ANEXO A. ESPECIFICACIONES Y ESTRUCTURAS DE TRAMAS DEL IEEE 805.15.4

A.1 CAPA FISICA

La capa física del 802.15.4 está separada en dos subcapas: *PHY data service* y *PHY management* que son las encargadas de transmitir y recibir mensajes a través del medio fisico.

Algunas características globales de la capa física son el control del transceptor de radio, calidad del enlace (LQI), selección de canal, detector de energía(ED), detección de portadora (CCA) para su uso en CSMA-CA a nivel MAC etc.

El estándar define dos opciones de transmisión según la banda de frecuencia utilizada (868/915 MHz y 2450 MHz), ambos están basados en el DSSS1. Como se puede observar en la Figura A.1 se dispone de distintas velocidades de transmisión dependiendo la frecuencia que se utilice, lo que conlleva que a mayor frecuencia mayor velocidad pero menor área de cobertura debido a la atenuación de la señal a frecuencias elevadas.

PHY	Frequencia(MHz)	Spreading parameters		Data Parameters		
		Chip Rate (Kchip/s)	Modulation	Aplicaciones	Ksimbols/s	2^n
868/915	868-868,6	300	BPSK	20	20	1
	902-928	600	BPSK	40	40	1
2450	2400-2483,5	2000	O-QPSK	250	62,5	4

Figura A.1: Velocidades de transmisión

En la banda de 2,4GHz, que es la más eficiente en cuanto al uso del ancho de banda, disponemos de un total de 16 bandas o canales de 2 MHz con un distancia entre canales de 5MHz para minimizar interferencia por canal adyacente tal y como se muestra en la Figura A.2.

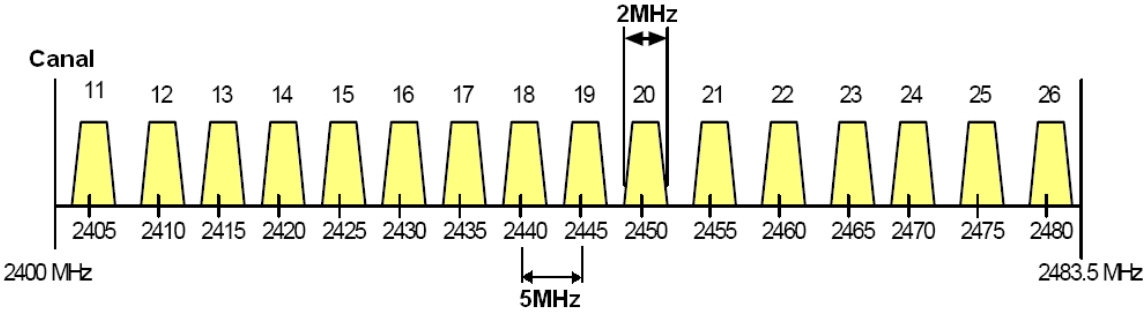

Figura A.2 División espectral de canales en la banda de 2.4 GHz

La unidad de datos a nivel físico (PPDU) que se ilustra en la figura A.3 está compuesta por tres partes:

SHR: permite a un dispositivo receptor sincronizarse para poder leer bien la información contenida en la PPDU, también indica el final de trama ya que la trama puede tener una longitud variable.

PHR: indica la longitud de información ya que esta puede ser variable como hemos comentado anteriormente.

PSDU: es la información de la trama a nivel físico.

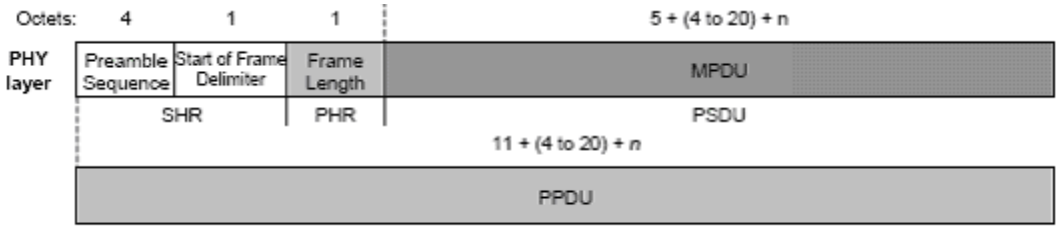

Figura A.3: Estructura Paquete PPDU

- El **preámbulo** es una secuencia de 32 ceros consecutivos. El campo preámbulo se utiliza para la sincronización con el mensaje entrante. El campo de preámbulo está compuesto de 32 ceros binarios

- **SFD:** El SFD es un campo de 8 bit que indica el final de la sincronización (el preámbulo) y el principio de los datos de paquete. El SFD tiene el formato ilustrado en la Figura A.4.

Bits: 0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	0	0	1	0	1

Figura A.4: Start delimiter.

- **Frame Length:** Este campo tiene 7 bits de longitud y especifica el número total de octetos contenidos en el PSDU. Es un valor comprendido entre 0 y *aMaxPHYPacketSize* (127). La figura A.5 resume el tipo de *payload* dependiendo del valor contenido en este campo.

Frame length values	**Payload**
0-4	Reserved
5	MPDU (Acknowledgment)
6-7	Reserved
8 to *aMaxPHYPacketSize*	MPDU

Figura A.5: Valores del campo Frame Lenght.

- **PDSU:** El campo de PSDU tiene una longitud variable y lleva los datos del paquete PHY. Para todos los tipos de paquete de longitud cinco octetos o mayor que siete octetos, el PSDU contiene el marco de subcapa de MAC (p. ej., MPDU).

Capa MAC

La capa de control de acceso al medio (MAC) proporciona dos servicios: MAC *data service* y MAC *management service*. Ambos servicios interactúan en la capa MAC y permiten la transmisión y recepción de unidades de datos, MPDU[25], a través del servicio de que proporciona la capa fisica. Las funciones más relevantes son:

- Generar tramas beacon en el caso de un *PAN coordinator* y que el resto de nodos se sincronicen al ritmo de las tramas beacon.
- Mecanismo de acceso al medio CSMA-CA.
- Asociación o desasociación a una PAN.
- Funciones de seguridad (encriptación AES).
- Mecanismos de fiabilidad entre nodos (ACK's).

La ventaja de este nivel MAC respecto al de otros estándares es que tan solo se dispone de 21 primitivas de servicio o comandos, lo que redunda en un hardware más sencillo y más barato de fabricar.

El formato de marco de MAC está compuesto por un MHR[26], un MAC payload, y un MFR[27]. Los campos del MHR aparecen en un orden fijo, sin embargo, los campos de direcciones pueden no estar incluidos en todos los marcos. El formato MAC general es el ilustrado en la figura A.6.

<table>
<tr><th>Octets: 2</th><th>1</th><th>0/2</th><th>0/2/8</th><th>0/2</th><th>0/2/8</th><th>variable</th><th>2</th></tr>
<tr><td rowspan="2">Frame control</td><td rowspan="2">Sequence number</td><td>Destination PAN identifier</td><td>Destination address</td><td>Source PAN identifier</td><td>Source address</td><td rowspan="2">Frame payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td></tr>
<tr><td colspan="6">MHR</td><td>MAC payload</td><td>MFR</td></tr>
</table>

Figura A.6: Formato general de MAC.

[25] MPDU – MAC Protocol Data Unit
[26] MHR – MAC Header
[27] MFR – MAC Frame

- **Frame Control:** Su tamaño es de 16 bits y contiene la información que define el tipo de trama, los campos de direcciones y otros flags de control. El formato de este campo se puede ver en la figura A.7.

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame type	Security enabled	Frame pending	Ack. request	Intra-PAN	Reserved	Dest. addressing mode	Reserved	Source addressing mode

Figura A.7: Formato del campo Frame control.

- **Frame type:** Indica el tipo de trama y debería contener uno de los valores no reservados en la figura A.8.

Frame type value $b_2\, b_1\, b_0$	Description
000	Beacon
001	Data
010	Acknowledgment
011	MAC command
100–111	Reserved

Figura A.8: Valores del campo Frame Type.

- **Security Enabled:** Si está puesto a 1 significa que la encriptación está habilitada y si está a 0 indica que no está encriptado.

- **Frame Pending:** Si está fijado a 1 indica que quedan tramas pendientes, 0 en caso contrario.

- **Ack Request:** Si está a 1 indica que necesita confirmación. En caso contrario no se espera confirmación alguna.

- **Intra-PAN:** Este subcampo ocupa 1 bit y específica si el frame MAC debe de ser enviado dentro de la misma PAN (intra-PAN) o a otra PAN (inter-PAN). Si está puesto a 1 y tanto la dirección de destino como la fuente está presente entonces el frame no contendrá el campo identificador *source PAN*. Si está a 0 y tanto direcciones destino como direcciones de la fuente están presentes, la trama contendrá los campos origen y destino PAN

- **Destination Addresing Mode:** Su longitud es de 2 bits y contendrá uno de los valores catalogados en la figura A.9.

Si este subcampo es igual a cero y el subcampo *Frame Type* no especifica que esta trama sea de tipo *acknowledgment* o *beacon,* el campo *source addressing mode* debería ser distinto de cero, implica que la trama está dirigida al coordinador PAN con el identificador PAN especificado en el campo *source PAN.*

- **Source Addresing Mode:** Su longitud es de 2 bits y contendrá uno de los valores catalogados en la figuraA.9.

Si este subcampo es igual a cero y el subcampo *Frame Type* no especifica que este frame es un *acknowledgment* o un *beacon,* el campo *destination addressing mode* debería ser distinto de cero, esto implica que el frame ha sido originado desde el coordinador PAN con el identificador PAN especificado en el campo *destination PAN.*

Addressing mode value $b_1\ b_0$	Description
00	PAN identifier and address field are not present.
01	Reserved.
10	Address field contains a 16 bit short address.
11	Address field contains a 64 bit extended address.

Figura A.9: Posibles valores de los campos *destination y source addresing mode*

- **Sequence Numer:** Este campo tiene 8 bits de longitud y especifica un identifcador de secuencia único para cada trama.

- **Destination PAN identifier:** Su tamaño es de 16 bits y especifica el identifcador PAN único del destino previsto de la trama. Un valor en este campo de 0xffff representará el identificador PAN *broadcast,* que será aceptado como un identificador PAN válido por todos los dispositivos que en ese momento escuchan el canal.

Este campo estará incluido en la trama MAC sólo si el subcampo *destination addressing mode* del *Frame Control es* distinto de cero.

- **Destination Address:** La longitud de este campo puede ser de 16 o 64 bits según el valor especificado en el subcampo *destination addressing mode* del campo *Frame Control*, y especifica la dirección de destino prevista de la trama. Un valor de 0xffff en este campo representará una dirección corta de *broadcast* que será aceptada como válida por todos los dispositivos que actualmente escuchan el canal.

- **Source PAN identifier: Destination PAN identifier:** Su tamaño es de 16 bits y especifica el identifcador PAN único del creador de la trama. Este campo estará incluido en la trama MAC sólo si los subcampos *sourece addressing mode* e *intra-PAN* del campo *frame control* son distintos de cero y cero respectivamente.

El identificador PAN de un dispositivo se determina inicialmente durante la asociación de un PAN, pero puede cambiarse después de una resolución de conflicto de identificador PAN.

- **Source Address:** La longitud de este campo puede ser de 16 bits o de 64 bits, según el valor especificado en el subcampo *destination addresing mode* del campo *frame control,* y especifica la dirección del creador de la trama. Este campo estará incluido en la trama MAC sólo si el subcampo *source addressing mode* del frame control es distinto de cero.

- **Frame Payload:** Tiene una longitud variable y contiene la información específica de cada tipo de trama individual. Si el subcampo *securite enabled* está a 1, el payload está protegido de acuerdo a la encriptación seleccionada.

- **FCS:** Su tamaño es de 16 bits y contiene un código de comprobación redundancia cíclica. El FCS se calcula sobre el MHR y el MAC *payload*.

Formato individual de los tipos de tramas

El estándar 802.15.4 define 4 tipos de trama: trama beacon, trama de datos, trama ACK y trama de comandos MAC.

Trama beacon:

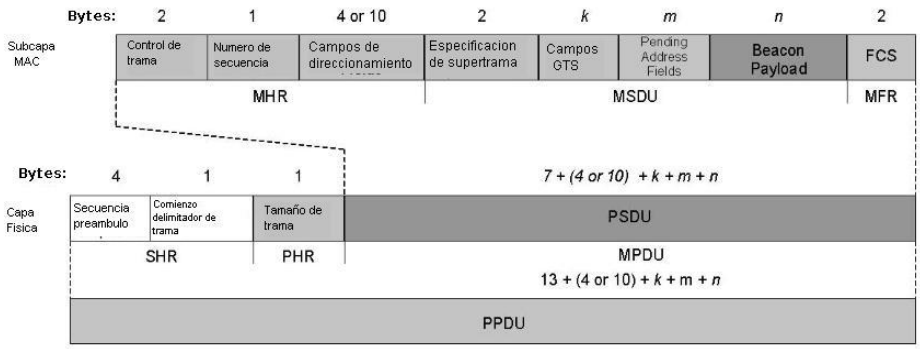

Figura A.10: Trama beacon.

Especificación de supertrama: Este campo especifica diversos parámetros referentes a la supertrama, tales como tiempo de supertrama, quien transmite la trama beacon o si permite la asociación de nuevos nodos. Nos da información para que el resto de nodos sepan cuando pueden transmitir.

Campos GTS: Aquí se especifica que nodo quiere latencia mínima y posición que ocupará dentro de la trama CFP.

Pending Address Fields: A modo informativo nos dice desde que direcciones se va a transmitir o recibir tramas.

Beacon payload: Este campo se utiliza para llevar información de capas superiores o información encriptada en caso de usar el algoritmo de encriptación AES establecido en el estándar.

Trama de datos:

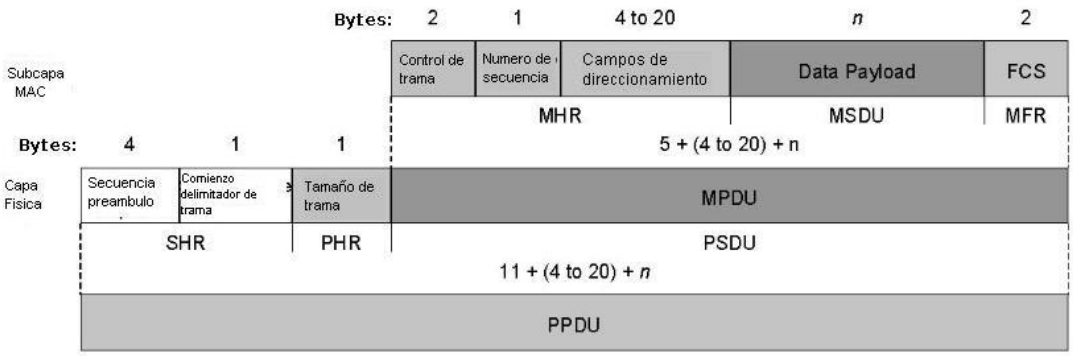

Figura A.11: Trama de datos

Data payload: Únicamente contendrá la información indicada por capas superiores.

Trama ACK:

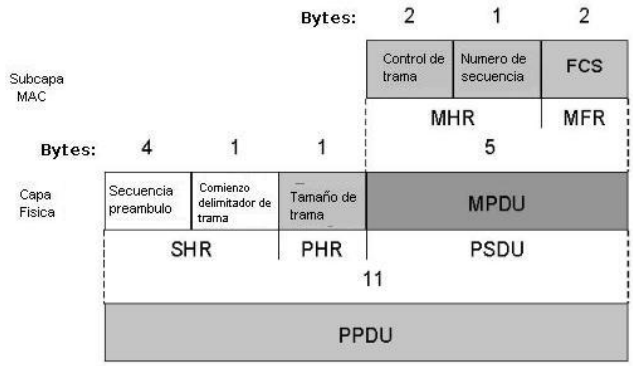

Figura A.12: Trama ACK

La trama ACK no lleva ningún campo específico, únicamente contiene el emisor que la trama de datos o que la trama de comandos MAC espera recibir desde la dirección emisora. El receptor envía una trama ACK para confirmar la llegada.

Trama de comando MAC:

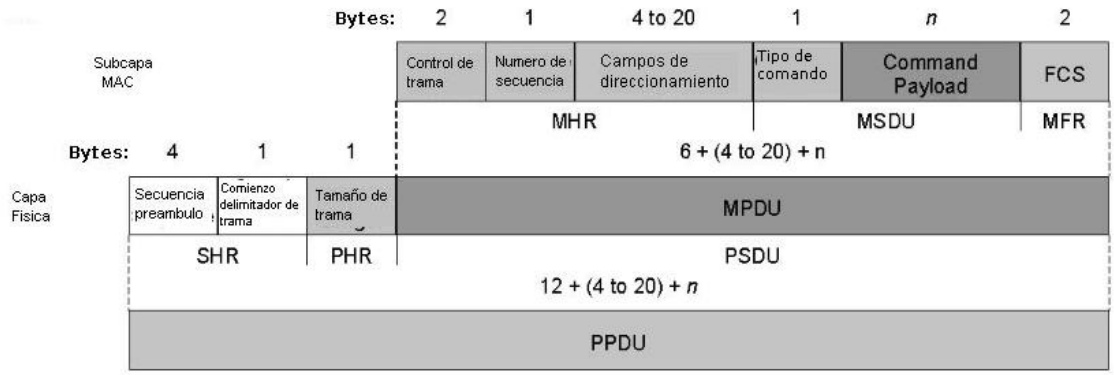

Figura A.13: Formato de la trama MAC

La trama de comandos MAC nos permite llevar a cabo todas las funcionalidades primitivas que define el estándar 802.15.4 tales como asociación, desasociación, resincronización con *PAN's coordinators*, solicitudes de latencia con QoS (GTS), etc...

Tipo de comando: Indica que tipo de comando solicita entre un total de nueve distintos.

Command payload: Su longitud es variable en función del comando utilizado.

ANEXO B. Repertorio de instrucciones MSP430

Conjunto de instrucciones del MSP430																
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Instruccción**
0	0	0	1	0	0	opcode			B/W	As		register				Single-operand arithmetic
0	0	0	1	0	0	0	0	0	B/W	As		register				**RRC** Rotate right through carry
0	0	0	1	0	0	0	0	1	0	As		register				**SWPB** Swap bytes
0	0	0	1	0	0	0	1	0	B/W	As		register				**RRA** Rotate right arithmetic
0	0	0	1	0	0	0	1	1	0	As		register				**SXT** Sign extend byte to word
0	0	0	1	0	0	1	0	0	B/W	As		register				**PUSH** Push value onto stack
0	0	0	1	0	0	1	0	1	0	As		register				**CALL** Subroutine call; push PC and move source to PC
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	**RETI** Return from interrupt; pop SR then pop PC
0	0	1	Condition			10-bit signed offset										Conditional jump; PC = PC + 2×offset
0	0	1	0	0	0	10-bit signed offset										**JNE/JNZ** Jump if not equal/zero
0	0	1	0	0	1	10-bit signed offset										**JEQ/JZ** Jump if equal/zero
0	0	1	0	1	0	10-bit signed offset										**JNC/JLO** Jump if no carry/lower
0	0	1	0	1	1	10-bit signed offset										**JC/JHS** Jump if carry/higher or same
0	0	1	1	0	0	10-bit signed offset										**JN** Jump if negative
0	0	1	1	0	1	10-bit signed offset										**JGE** Jump if greater or equal
0	0	1	1	1	0	10-bit signed offset										**JL** Jump if less
0	0	1	1	1	1	10-bit signed offset										**JMP** Jump (unconditionally)
opcode				**Source**			**Ad**		**B/W**	**As**		**destination**				**Two-operand arithmetic**
0	1	0	0	Source			Ad		B/W	As		destination				**MOV** Move source to destination
0	1	0	1	Source			Ad		B/W	As		destination				**ADD** Add source to destination
0	1	1	0	Source			Ad		B/W	As		destination				**ADDC** Add source and carry to destination
0	1	1	1	Source			Ad		B/W	As		destination				**SUBC** Subtract source from destination (with carry)
1	0	0	0	Source			Ad		B/W	As		destination				**SUB** Subtract source from destination

Conjunto de instrucciones del MSP430									
1	0	0	1	Source	Ad	B/W	As	destination	**CMP** Compare (pretend to subtract) source from destination
1	0	1	0	Source	Ad	B/W	As	destination	**DADD** Decimal add source to destination (with carry)
1	0	1	1	Source	Ad	B/W	As	destination	**BIT** Test bits of source AND destination
1	1	0	0	Source	Ad	B/W	As	destination	**BIC** Bit clear (dest &= ~src)
1	1	0	1	Source	Ad	B/W	As	destination	**BIS** Bit set (logical OR)
1	1	1	0	Source	Ad	B/W	As	destination	**XOR** Exclusive or source with destination
1	1	1	1	Source	Ad	B/W	As	destination	**AND** Logical AND source with destination (dest &= src)

Las instrucciones son 16 bit seguidas de hasta dos palabras de extensión de 16 bits. Hay cuatro modos de direccionamiento, especificados por el bit 2 como campo. Algunas versiones especiales pueden ser construidas usando R0, y otros modos a parte del acceso directo al registro usando R2 (el registro de estado) y R3 (el generador constante) son interpretados especialmente.

Los modos de direccionamiento indexado añaden una palabra de extensión de 16 bits a la instrucción.

Modos de direccionamiento del MSP430			
As	**Register**	**Syntax**	**Description**
00	n	**Rn**	**Register direct. The operand is the contents of Rn.**
01	n	**x(Rn)**	**Indexed. The operand is in memory at address Rn+x.**
10	n	**@Rn**	**Register indirect. The operand is in memory at the address held in Rn.**
11	n	**@Rn+**	**Indirect autoincrement. As above, then the register is incremented by 1 or 2.**
Addressing modes using R0 (PC)			
01	0 (PC)	**LABEL**	**Symbolic. x(PC) The operand is in memory at address PC+x.**
11	0 (PC)	**#x**	**Immediate. @PC+ The operand is the next word in the instruction stream.**
Addressing modes using R2 (SP) and R3 (CG), special-case decoding			
01	2 (SR)	**&LABEL**	**Absolute. The operand is in memory at address x.**
10	2 (SR)	**#4**	**Constant. The operand is the constant 4.**
11	2 (SR)	**#8**	**Constant. The operand is the constant 8.**
00	3 (CG)	**#0**	**Constant. The operand is the constant 0.**
01	3 (CG)	**#1**	**Constant. The operand is the constant 1. There is no index word.**
10	3 (CG)	**#2**	**Constant. The operand is the constant 2.**
11	3 (CG)	**#-1**	**Constant. The operand is the constant -1.**

Las instrucciones generalmente consumen un ciclo (de reloj) por palabra adquirida o almacenada, por lo que la duración de las instrucciones va desde 1 ciclo para instrucciones de registro a registro a 6 ciclos para una instrucción con los dos operandos indexados.

Operaciones de transferencia al contador de programa son legales y conllevan saltos. Los retornos de una subrutina por ejemplo, se implementan así: MOV @SP+,PC. En instrucciones con dos operandos, solo hay un bit "Ad" para especificar el modo de direccionamiento del destino, por lo que solo los modos 00 (directamente a registro) y 01 (indexado) están permitidos. Si tanto el origen como el destino son indexados, la palabra de extensión del origen va primero.

Cuando R0 (PC) o (R1) SP son usados con el modo de acceso de autoincremento, siempre son incrementados en dos. Otros registros (desde R4 a R15) son incrementados en el tamaño del operador, 1 o 2 bytes.

El registro de estado contiene cuatro bits de estado aritmético, una tabla de interrupciones global, y 4 bits que deshabilitan varios relojes para entrar en modo de bajo consumo. Cuando se atiende una interrupción, el procesador almacena el registro de estado en la pila y limpia los bits de bajo consumo. Si el vector de interrupción no modifica el registro de estado almacenado, volver de la interrupción, rehabilitará el modo de bajo consumo que estuviese habilitado previamente.

Glosario de términos

CPU - Central Processing Unit
RISC – Reduced Instruction Set Computer
MAB – Memory Adrress Bus
MDB – Memory Data Bus
JTAG - Joint Test Action Group
USART - Universal Synchronous Asynchronous Receiver Transmitter
LED - Light-Emitting Diode
FHSS - Frequency Hopping Spread Spectrum
MAC – Medium Access Control
TCP – Transmission Control Protocol
MSPSim – MSP Simulator
COOJA – COntiki Os Java simulator
PHY – Physical Layer
CSMA-CA – Caririer Sense Multiple Access with Collision Avoidance
DSSS1 - *Direct Sequence Spread Spectrum*
SHR *– Syncronization Header*
PHR *– PHY Header*
PSDU *– PHY Service Data Unit*
MPDU *– MAC Protocol Data Unit*
MSDU – MAC Service Data Unit
PAN – Personal Area Network
Payload – Datos encapsulados
MHR – MAC Header
ACK – Acknowledgment
QoS – Quality of service
PWM – Pulse Width Modulation
GTS – Guaranteed Time Slot

Palabras clave

- Redes de sensores
- Simulador
- ECG
- MSP430
- CC2420
- MSPSim
- COOJA
- SHIMMER
- Facultad de Informática
- Consumo potencia

Los alumnos Alejandro Asensio González, Arturo Miguel Núñez y Javier Pascual García como autores de este proyecto, **autorizan** a la Universidad Complutense de Madrid a difundir y utilizar, mencionando expresamente a sus autores, con fines académicos, no comerciales tanto esta memoria como el código generado a lo largo de este proyecto.

Firmado:

Alejandro Asensio González

Arturo Miguel Núñez

Javier Pascual García